UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Codificación y procesamiento de series temporales multivariantes atendiendo a su estructura temporal

Jorge Muñoz Aguado

Tutor: Pablo Varona Martínez

Julio 2020

Codificación y procesamiento de series temporales multivariantes atendiendo a su estructura temporal

AUTOR: Jorge Muñoz Aguado

TUTOR: Pablo Varona Martínez

Grupo de Neurocomputación Biológica

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2020

Resumen (Castellano)

El procesamiento de series temporales, es decir, el análisis de datos con una organización temporal relevante para el problema que se quiere resolver, aparece en múltiples aplicaciones tecnológicas. Lo encontramos por ejemplo en la predicción de indicadores económicos, en el comportamiento de valores biomédicos, en el estudio de la transmisión de enfermedades infecciosas como el COVID-19, o en el análisis del lenguaje de signos.

En este trabajo fin de grado se propone el estudio de la influencia de distintos tipos de codificaciones de entrada y técnicas de preprocesamiento de datos en la eficiencia de la clasificación de las series temporales multivariantes. Todo ello se analizará sobre modelos de aprendizaje automático de redes neuronales como redes multicapa, redes recurrentes *(RNN)* o redes convolucionales *(CNN).*

El estudio utilizará series temporales del lenguaje de signos resultantes de analizar la posición de la mano en distintos instantes de tiempo. Este ejemplo representativo nos permitirá probar distintos modelos y evaluar los resultados de cada uno de ellos.

Tras estudiar todas las técnicas propuestas, se pondrán en práctica sobre dos conjuntos de datos correspondientes al lenguaje de signos. Se analizarán los resultados con el fin de indicar qué codificaciones, técnicas de procesamiento de datos y modelos de redes neuronales mejoran la representación de la estructura temporal, con el fin de obtener una mayor precisión en la clasificación de series temporales multivariantes.

Además, se pondrá de manifiesto la relevancia de las series temporales en el campo de la inteligencia artificial y cuáles son las posibles aplicaciones que se pueden derivar de este estudio.

Palabras clave (Castellano)

Serie temporal, codificación, preprocesamiento, inteligencia artificial, red neuronal, lengua de signos, red convolucional, red recurrente, clasificación.

Abstract (English)

The processing of time series, i.e., of data containing a relevant time organization for the problem to be solved, appears in multiple technological applications. We can find it for instance in the prediction of economic indicators, in the behavior of biomedical values, in the study of the transmission of infectious diseases such as COVID-19, or in the analysis of sign language.

In this final degree project, the study of the influence of different types of input coding, as well as data preprocessing techniques, on the efficiency in classification of multivariate time series it is proposed. All these details will be analyzed on machine learning models of neural networks such as multilayer networks, recurrent networks *(RNN)* or convolutional networks *(CNN)*.

The study will use time series of sign language resulting from analyzing the position of the hand at different moments over time. This representative example will allow us to test different models and evaluate the results of each of them.

Once all the proposed techniques are studied, they will be put into practice on two data sets corresponding to sign language. The results will be analyzed in order to indicate which encodings, data processing techniques and neural network models improve the representation of the time structure, in order to obtain greater precision in the classification of multivariate time series.

In addition, we will bring out the relevance of multivariate time series in the field of artificial intelligence and we will discuss which are the possible applications that can be derived from this study.

Keywords (English)

Time series, encoding, preprocessing, artificial intelligence, neuronal network, sign language, convolutional neural network, recurrent neural network, classify.

Agradecimientos

Creo que todas las personas que pasan por nuestra vida influyen de alguna forma en ella, tanto las personas que nos han apoyado, como las que nos han criticado. Gracias a ellas he estudiado el grado que tanto deseaba desde que era un niño.

Aunque hay personas que han influido más que otras. En mi caso son mis padres, a los que tengo que agradecer todo. Gracias por haberme cuidado, gracias por haberme enseñado, gracias por haberme aguantado en momentos difíciles y muchas gracias por haber traído al mundo a otra personita que ha marcado mi vida: mi hermana, a la que quiero tanto.

Sin mi familia no sería el mismo. Mis tíos, mis abuelos, que tanto han confiado en mí, que tanto me han apoyado en los momentos difíciles. Gracias.

Y qué decir de mis amigos, con los que he pasado tantos buenos momentos. Las risas son importantes para ser feliz. Gracias.

Gracias a mis profesores por haberme ofrecido la oportunidad de estudiar lo que siempre me ha gustado. Sobre todo, gracias Nacho, gracias Rosa, por haberme guiado durante mis dos últimos años de colegio. Sin vosotros nunca habría descubierto mi interés por la informática.

Por último, me gustaría recordar una frase que para mí representa lo más bonito que tengo en esta vida: mi familia y mis amigos. Me recuerda que la vida es muy frágil y que tenemos que disfrutarla. Siempre hay sonreír, aunque pasemos por un mal momento, nunca olvidemos a quien queremos y quienes somos. Porque un día todo eso desaparecerá, todos esos momentos, todos esos recuerdos se borrarán para siempre.

All those moments will be lost in time, like tears in rain.

(Blade Runner (1982). Dirigido por Ridley Scott)

Gracias abuelo, te quiero.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

Las series temporales son un conjunto de observaciones con una estructura temporal que quedan registradas en distintos intervalos de tiempo [1], [2]. En otras palabras, son datos recogidos de forma sucesiva en el tiempo y que preservan la secuencia temporal en la que se producen. Estas observaciones se pueden componer de una o varias medidas simultáneas, recibiendo los nombres de *univariante* o *multivariante*, respectivamente, cada tipo de problema [3].

El lenguaje de signos es un lenguaje utilizado por personas con discapacidad auditiva para comunicarse. Ha habido estudios en el ámbito de la informática tratando de examinar los gestos que componen este lenguaje. El análisis automático del lenguaje de signos es un problema que aborda la clasificación automática de series temporales provenientes del seguimiento de los gestos realizados con las manos [4]–[7].

A continuación, plantearemos el alcance y las motivaciones de este trabajo de fin de grado en el contexto del procesamiento de series temporales sobre la lengua de signos.

1.1 *Motivación*

El tiempo es una variable que no en todos los problemas de procesamiento de datos se tiene en cuenta. En algunos no es necesario, en otros, considerarlo es complejo. El tiempo es una variable diferente a las demás; en la mayoría de los casos no se puede tratar como algo discontinuo y heterogéneo, ya que típicamente se registra en intervalos periódicos estructurando la secuencia de la información. El pasado es el antecedente del presente y, éste del futuro.

Debido a la complejidad del análisis de los datos respecto al tiempo, es un ejercicio menos común que, por ejemplo, el uso promediado de datos [8]. Sin embargo, asociar las secuencias de una estructura temporal a una funcionalidad específica puede ser útil en muchos tipos de aplicaciones. Por ejemplo, variables etiquetadas temporalmente permiten convertir un sistema caótico en uno analizable y predecible a corto plazo [9].

Existe un gran número de problemas que pueden hacer uso de las series temporales, como son algunos indicadores económicos [10], la predicción de efectos de fármacos y otros parámetros biomédicos [11] o la predicción de la transmisión y la evolución de enfermedades infecciosas, como por ejemplo el COVID-19 [12].

En particular me surge el interés de conocer en profundidad el uso y análisis de las series temporales, así como aprender a diseñar y examinar modelos y codificaciones de datos con distintas consideraciones sobre su estructura temporal. Todo esto se estudiará en este TFG sobre series temporales provenientes del análisis de lenguaje de signos.

Los conocimientos adquiridos se podrían extrapolar para crear aplicaciones con el fin de resolver problemas concretos. Por ejemplo, la detección de signos permitiría crear un traductor de lengua de signos optimizado que posibilitaría a las personas que requieren de este lenguaje para comunicarse poder hacerlo eficazmente con personas que no entienden los signos. Esto facilitaría a las personas con deficiencias auditivas o vocales una mayor integración al poder comunicarse por sí mismos para realizar acciones del día a día.

1.2 *Objetivos*

En esta tesis se estudiarán las codificaciones, técnicas de preprocesamiento de datos y modelos de redes neuronales aplicables a la clasificación de series temporales para enfatizar la importancia de la representación de la estructura temporal en cada tipo de problema. El objetivo es comparar las distintas técnicas estudiadas para determinar cuáles arrojan resultados de mayor precisión en un problema de series temporales, cuyos datos en este caso pertenecen al lenguaje de signos.

La interpretación de los movimientos de manos en el lenguaje de signos es compleja. Es un problema que requiere analizar la secuencia de posiciones que toman las manos en cada momento a lo largo del tiempo, para así interpretar qué gesto se está realizando.

Así pues, se comenzará analizando los datos, extrayendo sus características principales y, en caso de ser necesario, se aplicarán técnicas de preprocesamiento para mejorar la caracterización de su estructura temporal.

Más adelante, se estudiarán los modelos de redes neuronales que mejores resultados aportan y cómo se han de codificar los datos.

Finalmente, se realizarán numerosas pruebas y analizarán los valores obtenidos, explicando de forma justificada los motivos por los que se obtienen los distintos resultados.

Toda información extraída de documentos o artículos estará citada y referenciada de acuerdo con el estándar *IEEE*.

1.3 *Organización de la memoria*

A continuación, se numerarán los capítulos de esta tesis, acompañándolos con una pequeña descripción de qué se tratará en cada uno de ellos.

- ***Estado del arte***

  En este apartado se explicarán los conceptos fundamentales de las series temporales, así como toda la tecnología o información que será necesaria para comprender el diseño del trabajo y los resultados que arrojen las pruebas.

- ***Diseño y Desarrollo***

  Se explicará brevemente el diseño, la metodología y funcionamiento del código utilizado para la realización de pruebas. También se detallarán los datos utilizados para estas pruebas y los criterios de análisis de los resultados.

- ***Pruebas y Resultados***

  En este apartado se expondrán los resultados obtenidos tras la realización de las pruebas y se analizarán escrupulosamente con los criterios que se definirán para este propósito.

- ***Conclusiones y trabajo futuro***

  Finalmente, se comentarán de forma resumida los resultados obtenidos y se hará una valoración de la importancia de procesar, codificar y utilizar el modelo que más se adapte a la estructura temporal de los datos.

  Además, se hará una valoración de posibles trabajos futuros en el análisis de series temporales multivariantes.

2 Estado del arte

2.1 *Series Temporales*

Las series temporales son un conjunto de datos etiquetados temporalmente de acuerdo con el orden secuencial en el que se generan. Podemos representar una serie temporal como $S = <S_1, S_2, \ldots, S_N>$, donde S_X contiene un conjunto de datos asociados al tiempo [13], que puede ser interpretado como una señal. De esta forma, podemos denotar:

$$S_X =< (T_1 : D_1), \ldots, (T_M : D_M) >$$

En caso de que *S* tenga una única variable, $S = <S_1>$ se denomina *univariante*, en caso de tener más de una variable asociada al tiempo se le dará el nombre de *multivariante* [3] como se representa en la *figura 2*-1. Así, podemos decir que *S* es una composición de señales. Esta distinción será importante en el tratamiento de datos.

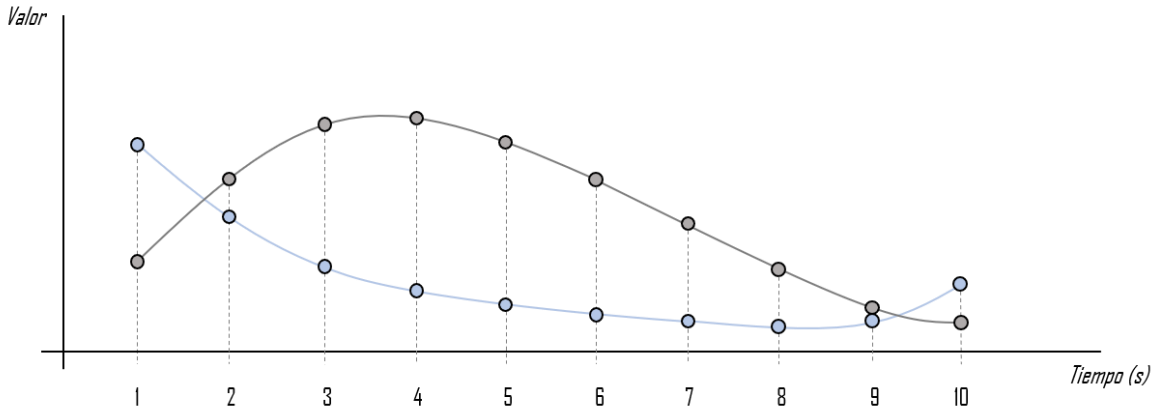

Figura 2-1: Ejemplo de dos señales que evolucionan en el tiempo

2.1.1 Estacionariedad de las series temporales

La estacionariedad es un factor determinante a la hora de clasificar o predecir las series temporales. Al analizar los datos conoceremos si tienen esta cualidad, lo que nos permitirá decidir qué representaciones son las más adecuadas en cada caso.

Hay varias de definiciones para determinar si una serie es *estacionaria*. La más estricta viene definida por las siguientes características, según el artículo [14]:

$$E(|X(t^2)|) < \infty$$

$$E\big(X(t)\big) = m$$

$$C\big(X(t_1), X(t_2)\big) = C\big(X(t_1 + \tau), X(t_2 + \tau)\big) = C(t_1 - t_2)$$

Donde *t*, t_1 *y* t_2 son instantes del tiempo, *X* la serie temporal, *m* es la media, τ es un offset, $E(\cdot)$ es el valor medio del conjunto y la función $C(\cdot)$ es la covarianza entre dos variables.

Por tanto, se puede afirmar que una serie estacionaria es aquella que tiene una covarianza constante y se le aplica la cualidad de *homocedasticidad*. Se dice que un conjunto de datos presenta *homocedasticidad* cuando la varianza es constante a lo largo de las observaciones.

Como se comenta en el mismo artículo [14], la estacionariedad de las series se suele conseguir con fuentes artificiales. Los datos naturales generalmente están influidos por variables aleatorias, por lo que no suelen cumplir las cualidades descritas anteriormente para ser considerados como series *estacionarias*.

Los datos *estacionarios* al tener una evolución sencilla son más fáciles de clasificar y predecir. Generalmente se pueden conseguir resultados con errores pequeños. En cambio, al introducir variables aleatorias a estos datos, pasan a ser *no estacionarios*, por lo que su clasificación es más compleja y las predicciones que se pueden realizar son a corto plazo ya que cuanto más a largo plazo se predice más error se acumulará.

En la *figura 2-2* se encuentran dos esquemas de señales correspondientes a una *serie estacionaria* y una *serie no estacionaria*, marcadas con las letras *a* y *b* respectivamente.

El problema del *random walk* es un buen ejemplo para ilustrar el inconveniente de predicción en los datos *no estacionarios*. Consiste en un caminante que, partiendo de un punto determinado, debe moverse a alguna posición adyacente de manera aleatoria. La probabilidad de error en la predicción se va acumulando a cada movimiento, ya que surgen nuevos posibles recorridos que puede haber tomado el algoritmo aleatorio.

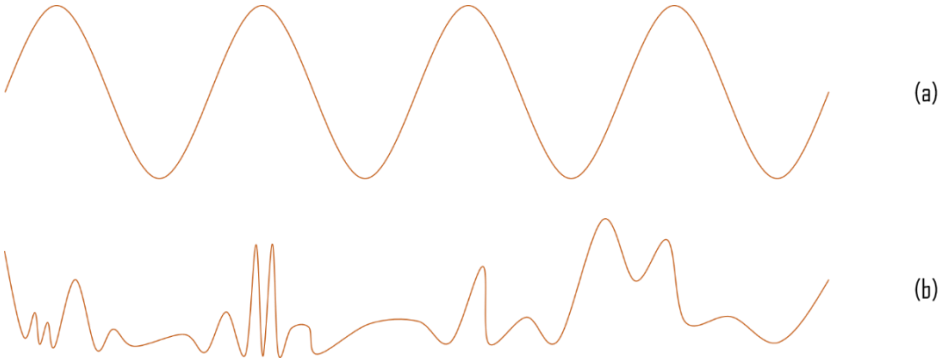

Figura 2-2: Ejemplo tipos de series temporales:
(a) una señal periódica, (b) una señal no estacionaria.

2.1.2 Clasificación

Clasificar es la acción de determinar a qué clase pertenece una serie temporal, donde entendemos *clase* como grupo o categoría. En el ámbito más general del aprendizaje automático sería decidir a qué clase pertenece un grupo de datos, donde puede haber una o más de estas clases.

El número de clases con las que cuente nuestro problema es un factor determinante a la hora de elegir un modelo adecuado. En las redes neuronales, por ejemplo, cada clase tendrá una salida asociada y la clase ganadora será la que corresponda a la salida con el mayor valor.

2.1.3 Predicción

Según la *RAE¸ Real Academia Española,* "predecir" consiste en: *"Anunciar con antelación algo que ha de suceder."* [15]. Por lo que, en el contexto de series temporales, se puede interpretar como: *"A partir de datos pasados estimar la evolución futura de una señal"*. En términos de inteligencia artificial podríamos decir que vamos a entrenar un modelo con los datos del pasado para poder predecir el futuro.

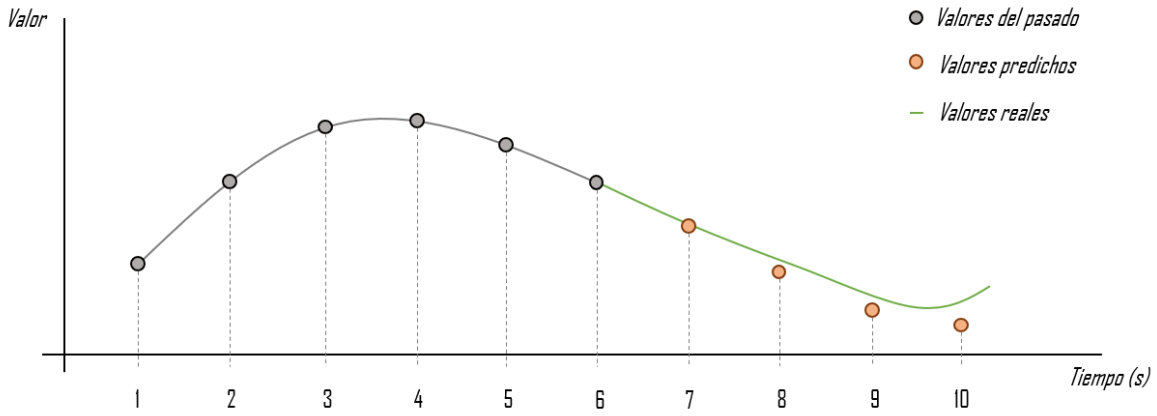

Figura 2-3: Esquema *forecast* con valores reales y predichos.

Datos del pasado			Dato predicho
4	5	6	7
5	6	7	8
6	7	8	9
7	8	9	10

Tabla 2-1: Ejemplo *forecast.*

La técnica de *forecast* es la más usada para predecir datos futuros. Consiste en predecir los datos futuros en función de los datos pasados [16], de manera recursiva. Los datos pasados podrán ser reales o datos predichos por el modelo. La *tabla 2-1* muestra un ejemplo de cómo actúa la técnica de *forecast* prediciendo los datos en una serie temporal *univariante*. Mientras en la *figura 2-3* se muestra un esquema con los datos predichos.

En el ejemplo expuesto en la *tabla 2-1* se puede observar que al predecir el dato en *t 10*, todos los datos usados para predecir el futuro eran datos anteriormente predichos. Como se observa en la *figura 2-3*, las predicciones recursivas por el método de *forecast* acumulan error ya que estiman los siguientes valores a partir de los ya predichos.

En las series temporales *multivariantes*, para aplicar la técnica de *forecast* se pueden utilizar una, varias o todas las series temporales del conjunto. Los modelos de aprendizaje automático y la codificación temporal que utilicen serán relevantes para predecir correctamente.

2.1.4 Análisis y Procesamiento de los datos

En cualquier problema de aprendizaje automático, antes de proceder al entrenamiento del modelo, típicamente se realiza un análisis y procesamiento de los datos. En este apartado se comentarán las metodologías más frecuentes en análisis de datos en series temporales, con especial énfasis en las que conservan la estructura temporal de las señales.

2.1.4.1 Análisis de los datos

En primer lugar, se deberá conocer el tipo de serie temporal que se está aplicando. Para más información recurrir al apartado ***2.1.1 Tipos de series temporales*** en el que se comentan las características de estacionariedad de las series temporales. Esta cualidad se puede extraer de un análisis matemático o, si se trata de un problema sencillo, de un análisis visual de los datos.

Para determinar si una serie es estacionaria, deberán identificarse los periodos de repetición. Este tipo de series son, en ocasiones, periódicas o cuasi-periódicas. Puede que no todos los datos de la serie cumplan las características, pero sí un subconjunto de éstos.

Los datos estacionarios se identifican con facilidad. Por ejemplo, la función *seno* representada en la *figura 2-2 (a)* en la que los datos se repiten en el tiempo. En este tipo de series, nunca se podrán generar valores superiores a los máximos, ni menores a los mínimos.

Los datos que no cumplan las características mencionadas para las series temporales estacionarias se considerarán *no estacionarios*. Estas series presentan complicaciones a la hora de predecir y clasificar, ya que la precisión puede ser muy baja si los datos utilizados durante el entrenamiento no representan adecuadamente la variabilidad de la señal. Para minimizar estos errores será necesario procesar los datos.

En la *figura 2-4* se representa una gráfica no estacionaria, con datos generados por el método de *random walk* comentado anteriormente. Estos datos no presentan ciclos de repetición y no existe una variabilidad estereotipada.

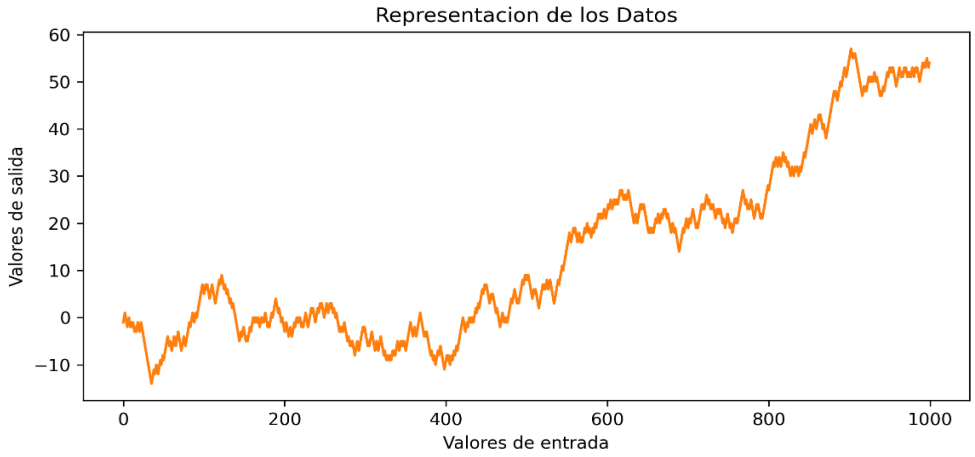

Figura 2-4: Ejemplo de serie temporal no estacionaria *random walk*.

2.1.4.2 Procesamiento de datos

Durante el procesamiento de los datos se pueden eliminar elementos que dificulten a los modelos de aprendizaje automático extraer las características más relevantes de los datos asociados a una serie temporal, así como mejorar la representación de sus características temporales para aumentar la precisión del modelo.

Dado que cada variable asociada al tiempo se puede interpretar como una señal, el procesamiento que se haga sobre los datos será procesamiento típicamente dirigido a señales. En esta sección se resumirá alguno de los métodos más extensamente utilizados. Para recabar más información sobre estos y otros métodos de procesamiento de series temporales se recomienda el artículo *"The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis"* [14].

2.1.4.2.1 Resolución de sampleo o granularidad

La palabra sampleo es un anglicismo que significa tomar porciones de una señal. En las series temporales entendemos sampleo o granularidad como la técnica a aplicar para modificar la resolución de los datos temporales.

Al recolectar o generar los datos, estos se toman en intervalos de tiempo. Si analizamos los datos podríamos observar que tienen una alta resolución, es decir, que la diferencia entre dos datos tomados en los tiempos *t* y *t+1* es muy pequeña. Aplicando la técnica de sampleo se podrían reducir algunos datos de forma intercalada, para que la resolución sea un poco más baja, sin perder la estructura temporal necesaria para resolver el problema a abordar. Esta técnica permite reducir el tamaño de los datos, por lo que puede contribuir a disminuir también el tiempo de entrenamiento.

En las series temporales *multivariadas* podemos encontrar variables con distintas resoluciones. Esto se puede deber a que los sistemas de medición toman valores en intervalos de tiempo diferente o, simplemente, a que unas variables evolucionan más lentamente o más rápidamente que otras. Al estar tomando datos durante el mismo intervalo de tiempo y tener resoluciones diferentes, el número de datos de cada variable puede ser dispar y esto puede dificultar su codificación.

Para solucionar este inconveniente se pueden aplicar distintas técnicas. El primer método consiste en reducir la resolución de las variables con mayor precisión. Esto implica que al reducir el número de datos pueden eliminarse valores representativos para el problema que se está abordando. En la *figura 2-5 (a)* se muestra un ejemplo esquemático de aplicar la reducción de la resolución.

Otra opción pasa por interpolar aquellas variables con un número de datos reducido. Estos nuevos datos se deben generar de forma que no produzca artefactos en la estructura temporal de los datos existentes. En la *figura 2-5 (b)* se muestra un ejemplo esquemático de cómo interpolar datos, donde los puntos son los datos y las líneas solo representaciones gráficas.

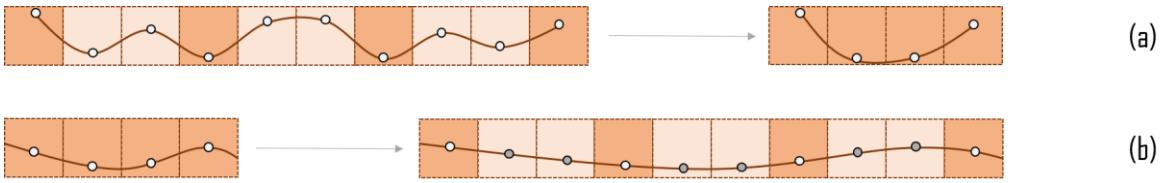

Figura 2-5: Tipos transformación.
(a) Ejemplo de resampleo de la señal perdiendo información.
(b) Ejemplo de interpolación de la señal.

2.1.4.2.2 Datos constantes

Un conjunto de datos también puede tener variables no dependientes del tiempo o con poca variabilidad durante el periodo de tiempo a analizar. Por ejemplo, los datos demográficos, que se actualizan en intervalos de tiempo muy grandes. A pesar de ser datos con poca variabilidad pueden ser utilizados para contextualizar el resto de las variables.

Los datos constantes se pueden plantear como un problema de granularidad, por lo que para solucionar este inconveniente se puede aplicar la técnica de interpolación explicada en el apartado anterior. En otros casos los valores constantes se pueden tratar como escalares para contextualizar el problema a resolver.

2.1.4.2.3 Normalización

La normalización trata de gestionar la magnitud de los datos de la serie temporal, mejorando así su presentación al algoritmo que tiene que operar sobre dicha serie. Existe un gran número de posibles técnicas que se pueden aplicar para mejorar la escala. La más común consiste en dejar la media de los datos a 0 y la desviación típica a 1. Esto se consigue restando la media y dividendo entre la desviación.

La finalidad de la normalización en estos casos es centrar los datos para que las variaciones entre máximos y mínimos se mantengan dentro de un rango. Esto en parte podría solucionar algunas de las dificultades que surgen al procesar las series temporales *no estacionarias*.

2.1.4.2.4 Filtrado

El filtrado es una de las técnicas más utilizadas en el procesamiento de señales. Consiste en aplicar un núcleo o *kernel* a la señal. Este núcleo es un vector de igual dimensión que los datos, cuyos valores varían en función de las necesidades [17].

Existen disposiciones de *kernel* estudiadas, cuyos resultados se conocen. Un ejemplo sería el filtrado de media y la derivada.

El filtrado de media consiste en suavizar la señal, realizando una media con el entorno de un dato. Es decir, se recalculará el dato *D(t)* haciendo una media de su valor con el de sus *k* vecinos, de tal forma que:

$$D(t) = \frac{D(t) + \sum_1^k (D(t-i) + D(t+i))}{2k+1}$$

En la *figura 2-6* se muestra el resultado de aplicar el filtrado de media a la señal presentada en la *figura 2-4*.

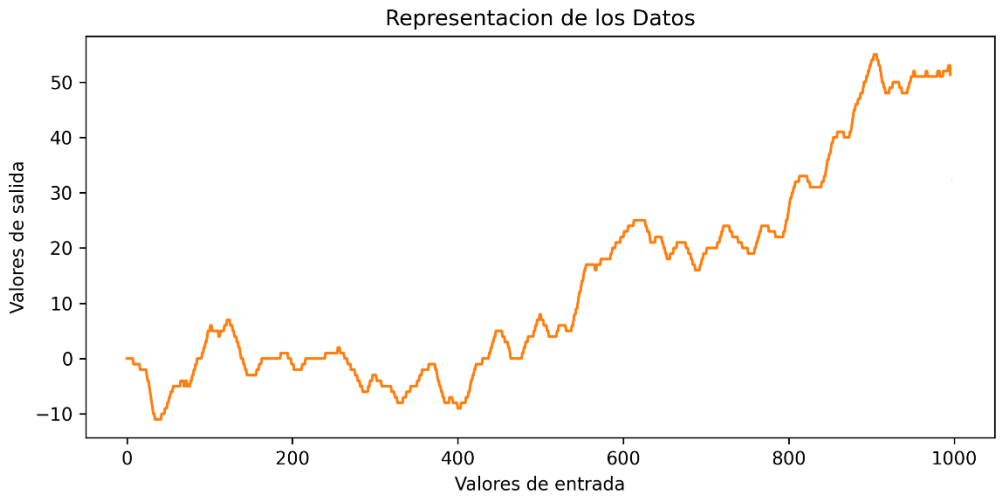

Figura 2-6: Ejemplo filtrado de media a datos *random walk*.

La derivada es una transformación que aporta información sobre la tendencia de cambio de las series temporales. Esta transformación es muy sencilla y se puede implementar como la diferencia entre datos adyacentes en el tiempo. Esto se puede conseguir aplicando un filtrado con el kernel *[-1, 1]*.

2.1.4.2.5 Espectrograma

El espectrograma se obtiene al dividir una señal o conjunto de señales en ventanas y calcular el espectro para cada una de ellas. El cálculo del espectro consiste en cuantificar las frecuencias que presenta la señal a lo largo del tiempo.

El análisis espectral proporciona una descomposición de señales complejas en otras más sencillas y este hecho puede ser muy útil en series *no estacionarias* [9].

2.1.4.2.6 Transformada Discreta de Fourier

La transformada discreta de Fourier es un método de transformación de señales diseñado por Joseph Fourier. Permite cuantificar la contribución de cada frecuencia en una señal.

Se basa en la siguiente fórmula donde *x* sería la señal, y *k* las frecuencias de la descomposición:

$$x(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}, \qquad k = 0, \dots, N-1$$

Para el tratamiento de los datos se suele usar la *transformada rápida de Fourier* o *FFT*, método que permite calcular la *transformada discreta de Fourier* o *DFT* de una forma rápida y eficiente.

2.1.4.2.7 Transformada de ondícula o wavelet

Esta transformación se basa en la convolución iterativa de señales, variando su longitud de onda, sobre los datos a transformar. Se aplica gracias a la siguiente fórmula:

$$W(a, b, X) = |a|^{-1/2} \int_{-\infty}^{\infty} X(t) \psi * \left(\frac{t-b}{a}\right) dt$$

Donde *a* define la escala, *b* es el factor de tiempo, *X* los datos.

Al igual que el espectrograma es capaz de descomponer la señal, aumentando el número de variables con las que contará la serie temporal.

2.1.5 Codificación de los datos

La codificación consiste en especificar cómo se van a presentar los datos al modelo de aprendizaje automático, que a su vez puede tener distintas estrategias para tener en cuenta la estructura temporal de las series temporales. Para la codificación, de unas variables originales se intenta extraer características que tengan relevancia en la clasificación y predicción. Es importante recalcar, que en el contexto de las series temporales conviene, en la mayoría de los problemas, conservar la estructura temporal y la secuencia de los datos.

2.1.5.1 Enventanado

El enventanado consiste en segmentar la señal en ventanas concatenadas o superpuestas. Esta codificación puede ayudar al modelo a secuenciar los datos temporales. En la *figura 2-7* se muestra un ejemplo esquemático de la codificación en ventanas.

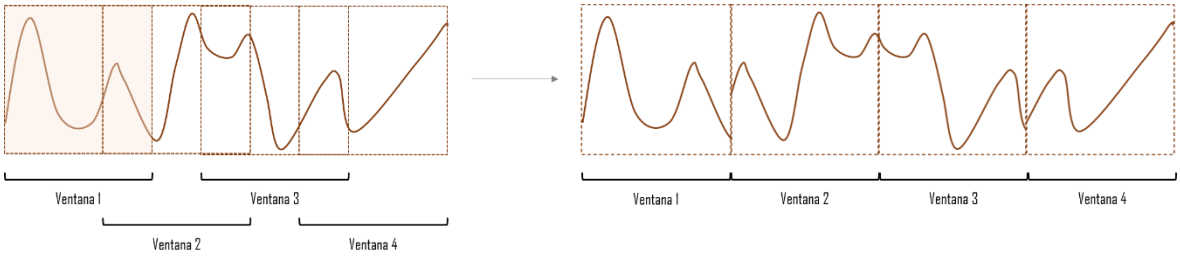

Figura 2-7: Ejemplo codificación en ventanas

2.1.5.2 Distancia entre series

La codificación propuesta consiste en utilizar la distancia entre series como datos de entrada al modelo de aprendizaje automático. La distancia se calcula entre la serie temporal que se vaya a codificar y unas series de referencia. Estas series de referencia pueden ser extraídas, por ejemplo, de los datos de entrenamiento [1]. Esto permite al modelo de aprendizaje automático predecir a qué clase corresponde, a partir de la distancia a la serie de referencia de esa clase.

La *figura 2-8* muestra un ejemplo esquemático de cómo se han de codificar las distancias, donde $D(S_X, S_Y)$ es la *distancia* entre S_X y S_Y, que se utilizan para decir la clase C de salida.

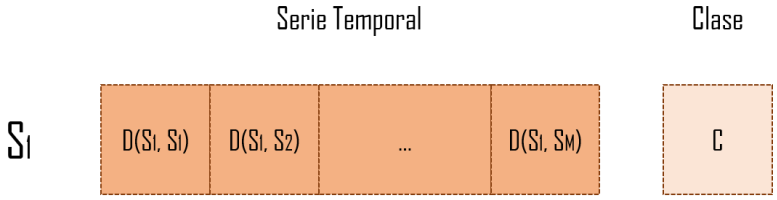

Figura 2-8: Resultado de la codificación de distancia entre series, para la serie S1. (Donde hay S_M series de referencia).

La distancia entre series se puede calcular mediante distintos métodos, por ejemplo, la distancia euclídea o la distancia coseno.

En general esta codificación suele arrojar buenos resultados, pero hay casos en los que puede no ser así. Por ejemplo, errores o latencias al medir los datos pueden producir un efecto conocido como *offset*. Este efecto consiste en que los datos pueden estar desplazados en el tiempo, tal y como muestra la *figura 2-9*, pero pertenecen a la misma clase. Esto puede provocar que la distancia entre datos de la misma clase sea muy amplia y no se consiga el efecto que se persigue.

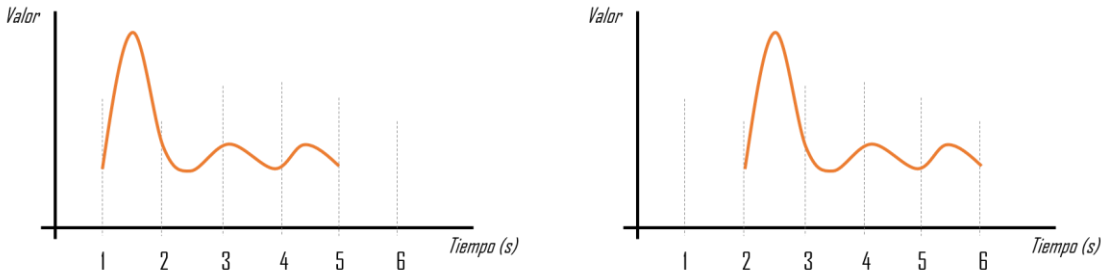

Figura 2-9: Datos desplazados en el tiempo, pero pertenecientes a la misma clase.

2.1.5.3 Shapelet

La codificación en *shapelets* se basa en extraer fragmentos característicos de los datos para cada clase. A cada una de estas secciones la conocemos por el nombre de *shap*. Al aplicar esta codificación se comprobará en qué porcentaje los datos a clasificar contienen estos fragmentos, reduciendo así la complejidad del problema y el tiempo de entrenamiento del modelo [18].

En la *figura 2-10* se muestra un ejemplo del funcionamiento de los *shapelets*, donde cada fragmento de color se corresponde con las partes más características de una clase.

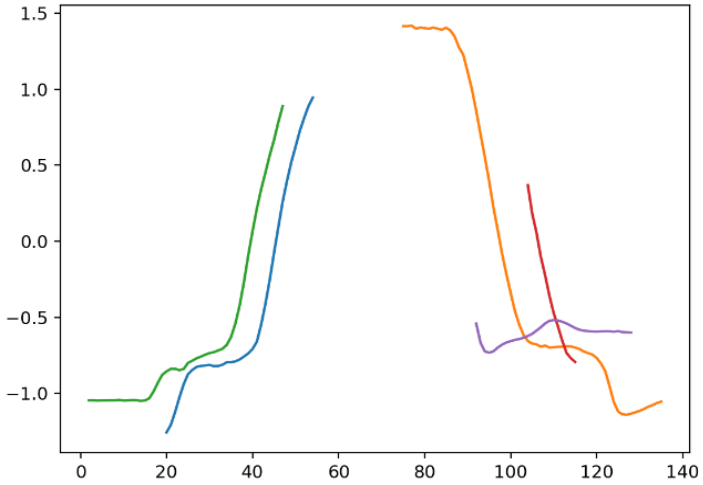

Figura 2-10: Ejemplo cálculo de los *shap*.

Los colores representan los shap o fragmentos representativos de una clase.

2.1.5.4 Representación de datos multidimensional

Esta codificación representa los datos de las series en varias dimensiones y por ello podrá modificar la disposición de las entradas del modelo. Mantiene las variables separadas e independientes, generando típicamente una matriz bidimensional de datos de entrada. Por ejemplo, en una serie temporal con 5 variables, donde cada variable tiene una resolución de 1000 datos, tendríamos una matriz 5x1000. Esto será útil para aquellos modelos que admitan entradas de más de una dimensión.

Una de las ventajas de la representación matricial es que se puede transponer fácilmente y de esta manera cambiar la forma en la que la información temporal se presenta al clasificador.

2.1.5.5 Representación de datos unidimensional

Cuando se trata un problema de series temporales *multivariantes* los datos también se pueden representar en formato unidimensional, típicamente concatenando las series en un único vector. Por ejemplo, en una serie temporal con 5 variables, donde cada variable tiene una resolución de 1000 datos, tendríamos un vector unidimensional de 5000 valores. Al igual que la codificación en matriz, esto condiciona la disposición de entrada de datos al modelo, facilitando generar modelos de una sola dimensión.

2.1.6 Sobreajuste

El *overfitting* o sobreajuste es un problema que se da cuando el modelo de aprendizaje automático obtiene una alta precisión en los datos de entrenamiento, pero muy baja precisión en los datos de test. Esto se debe a que el algoritmo aprende excesivamente los datos de entrenamiento y pierde la capacidad de generalizar.

Normalmente, hay un pequeño desajuste entre la precisión que se obtiene entre los datos de entrenamiento y test, lo cual no es un inconveniente. El problema surge cuando esta diferencia se agranda. Otra forma de medir el *overfitting* es comprobando si aumenta la diferencia entre el valor dado por la red y el real, sobre los datos de prueba.

Existen numerosas técnicas para evitar el sobreajuste como llevar la cuenta del error de clasificación durante el entrenamiento, aumentar el número de datos, aplicar la técnica de *Data Augmentation* [19] o en las redes neuronales impedir que algunos pesos varíen durante el entrenamiento, conocido como *Dropout* [20].

2.2 *Modelos*

En este apartado se detallarán algunos modelos relevantes en el problema de clasificación de series temporales, comentaremos brevemente su funcionamiento y la razón por la que han sido elegidos para el análisis de series temporales en este TFG. Además, se enumerarán las codificaciones posibles de los datos asociados al tiempo [21], [22].

2.2.1 Redes neuronales

Las redes neuronales, surgen de la idea de imitar las redes biológicas para solucionar problemas de inteligencia artificial. Estas redes artificiales han ido evolucionando y se han diseñado variaciones para cada tipo de problema.

Una característica interesante aplicada a las redes neuronales es que son *clasificadores genéricos.* Es decir, que son capaces de adaptarse a todos los problemas con facilidad. Aun así, cada tipo de red se especializa a través de su parametrización en un problema en particular.

Las redes están compuestas de neuronas conectadas entre sí. Cada neurona sumará sus entradas multiplicadas por los pesos de sus conexiones. La salida de la neurona será la salida de una *función de activación* cuya entrada es la suma pesada de la que hemos hablado. Existen diversas funciones de activación entra las que se encuentran la *sigmoide, relu* y *tanh,* entre otras [23].

2.2.1.1 Redes Neuronales Multicapa

Para el procesamiento de series temporales, las redes multicapa, típicamente *fully connected*, son redes neuronales que pueden contar con varias capas, cada una compuesta de una o más neuronas. Las neuronas de una misma capa están desconectadas entre sí, pero están conectadas con todas las de la capa siguiente y todas las de la anterior. En la *figura 2-11* se muestra un ejemplo gráfico de esta red neuronal.

Cada capa suele recibir un nombre característico. Las capas por donde entran los datos se denominan *capas de entrada*. Generalmente estas no se cuentan como una capa de la red ya que únicamente están compuestas por los datos. La última capa, llamada *capa de salida* será por donde obtenemos un resultado. Al resto de capas se las conoce como *capas ocultas*.

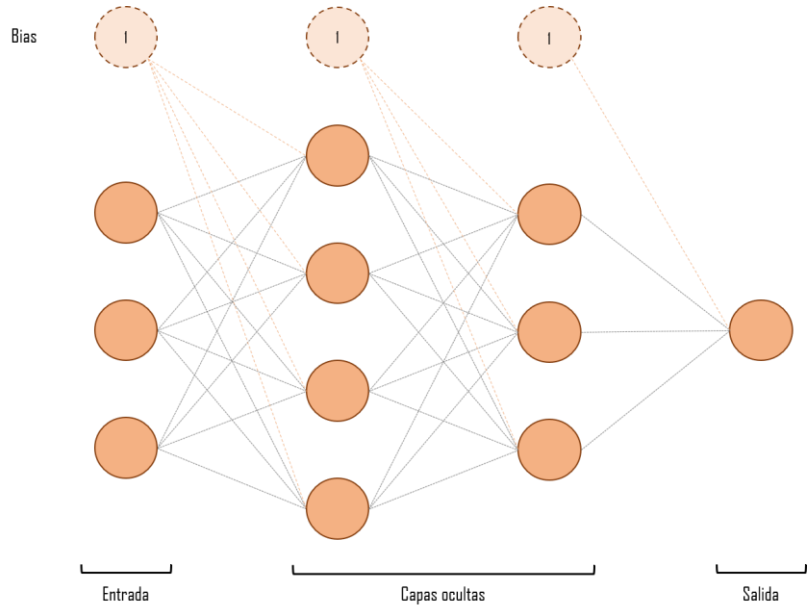

Figura 2-11: Red neuronal densa

2.2.1.2 Redes Recurrentes

Las redes recurrentes reciben este nombre ya que generan bucles de retroalimentación, donde los datos pueden perdurar por varias épocas. Las neuronas podrán ser de varios tipos y se realizarán distintos procedimientos dependiendo de las características de las mismas. Existen distintos tipos, pero solo nos vamos a centrar en las redes LSTM o *Long short-term memory*. En la *figura 2-12* se muestra un ejemplo esquemático de este tipo de red.

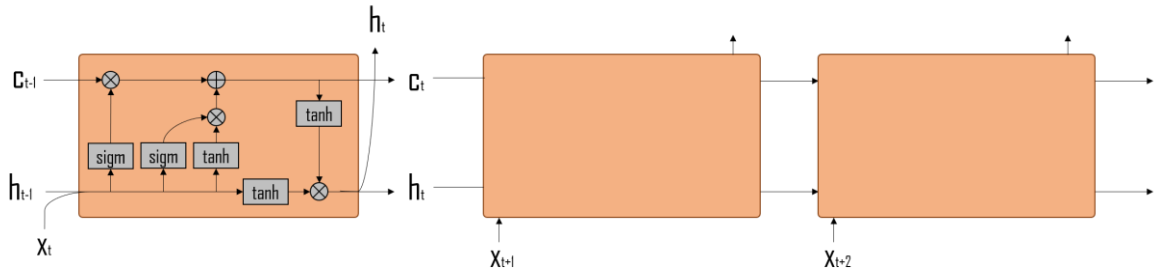

Figura 2-12: Red Neuronal Recurrente.

Este tipo de redes están especializadas en patrones secuenciales, generalmente asociados a un tiempo. Se basan en la posibilidad de poder recuperar información a partir de datos distorsionados. Este tipo de redes se usan para las series temporales ya que generalmente los datos de la misma clase presentan patrones secuenciales similares. Además, como se muestra en la *figura 2-15*, cada módulo está conectado con su modulo posterior, lo cual facilita la codificación del tiempo.

2.2.1.3 Redes Convolucionales

Las redes convolucionales son una evolución de las redes neuronales densas. Se suelen usar para tratamiento de imágenes y vídeo. Estas redes se basan en aplicar operaciones de filtrado, por lo que el preprocesamiento que hay que realizar a estas redes es menor que en el resto de los modelos.

Estas redes se componen de dos partes. La primera parte, convolucional, en la que cada capa de la red está constituida por una o más operaciones convolucionales, las cuales se aplicarán sobre la capa anterior. La segunda parte típicamente está compuesta por una red neuronal multicapa. La *figura 2-13* muestra un esquema de estas redes, donde se pueden diferenciar las partes comentadas.

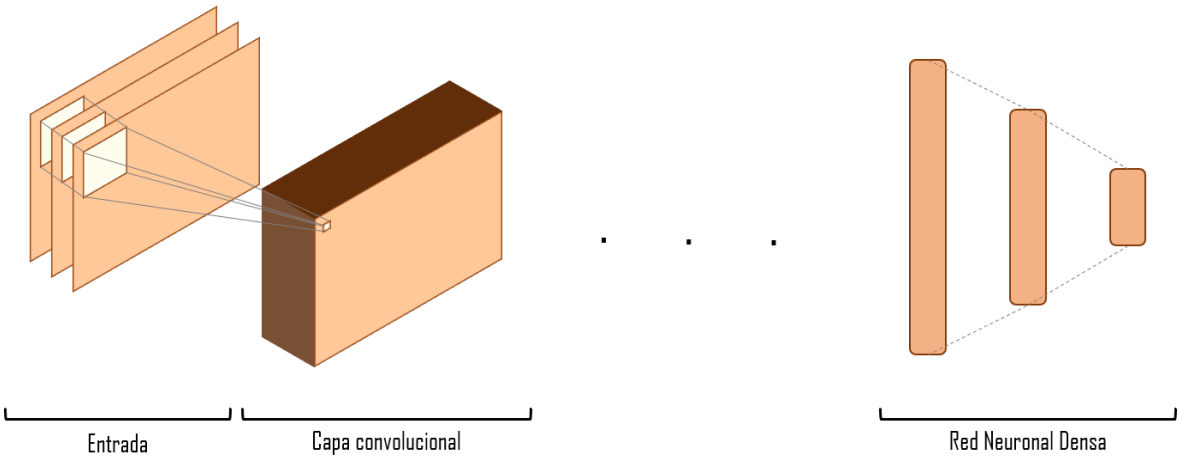

Figura 2-13: Red Neuronal Convolucional

Debido a estas características, la codificación de los datos se puede realizar de distintas maneras. En caso de contar con varias variables, cada dato se puede disponer en una subcapa de entrada o bien agrupar todos los datos en una única capa, pero con los datos típicamente dispuestos en dos o más dimensiones, de tal forma que cada línea sea un tipo de dato.

La decisión de elección de este tipo de redes para el análisis de series temporales se apoya en que, como ya se afirmó, las series temporales son señales que evolucionan en el tiempo y estas redes pueden representar adecuadamente su estructura temporal.

2.2.2 Métodos alternativos a las redes neuronales

Aparte de los modelos basados en redes neuronales, existen otros diseñados con conceptos diferentes. A continuación, se exponen algunos de los más relevantes usados para análisis de series temporales.

2.2.2.1 Vecinos próximos

El modelo de vecinos próximos o *KNN* se basa en determinar la clase de un dato en función de la clase mayoritaria entre los *K* vecinos más próximos. La distancia entre vecinos se puede calcular con distintas fórmulas como la *distancia coseno* o *distancia euclídea*. En la *figura 2-14* se representa un espacio bidimensional con dos tipos de datos. La línea gris se corresponde con la frontera en la que se decide a qué clase pertenece cada dato.

Este modelo presenta el inconveniente de que la clasificación es muy lenta, ya que para cada dato de *test* se debe calcular la distancia con todos los datos de *entrenamiento*.

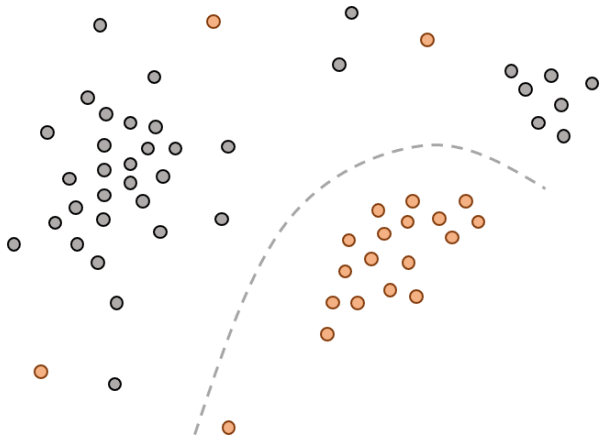

Figura 2-14: K-NN ilustrativo.

2.2.2.2 Random Forest

Este modelo consiste en crear varios árboles de decisión, entrenarlos y posteriormente escoger el valor de salida que más se repita. La idea es que cada árbol busque las características básicas de una variable dentro del conjunto de datos. Los *árboles de decisión* se basan en dividir los datos recursivamente, tratando de conseguir que el número de divisiones sean las mínimas.

Una codificación óptima para este modelo es la denominada *SMTS*, basada en una representación simbólica que puede preservar la estructura temporal [24].

3 Diseño y Desarrollo

3.1 *Diseño*

Para la realización de las pruebas aportadas en esta tesis se ha requerido de un programa que permite cargar datos de las series y manejar varios modelos de aprendizaje automático. El programa se basa en el diseño de clases, que se comentará a continuación.

3.1.1 Diseño de partes

Se ha optado por un diseño modular, en bloques independientes, basado en la orientación a objetos. Como se puede observar en la *figura 3-1* el programa se compone de tres bloques principales. Cada uno de ellos se explicará con más detalle en esta sección.

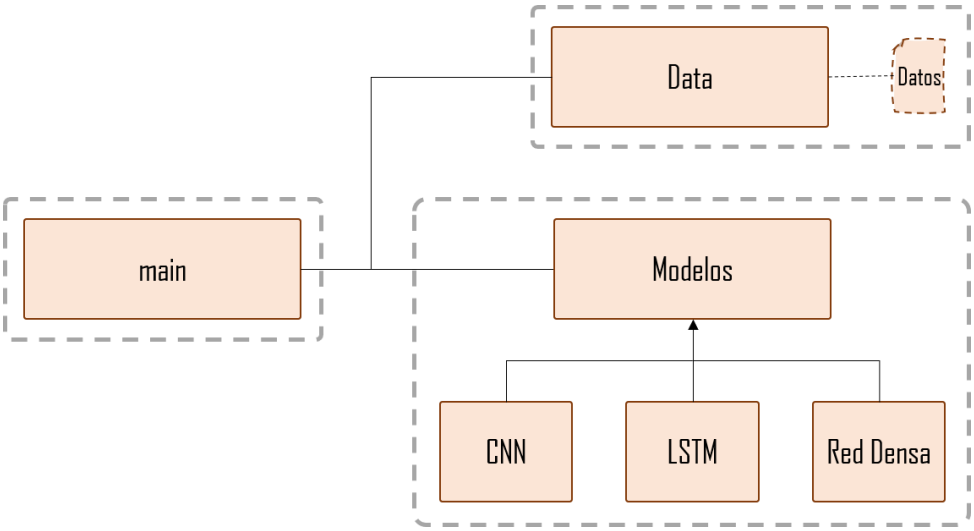

Figura 3-1: Esquema diagrama de clases

3.1.1.1 Módulo main

Este primer bloque es el correspondiente al programa principal, desde el que se llamarán al resto de clases y métodos implementados para las pruebas. Este bloque no aporta nueva funcionalidad, usará el resto de los módulos para generar los resultados.

3.1.1.2 Módulo de datos

Este módulo se encarga del manejo de los datos del *dataset*. Es decir, genera los datos a partir de funciones o carga los datos desde un fichero. También permite guardar los datos y crear las divisiones entre datos de entrenamiento y test.

Otra funcionalidad de este módulo es el análisis y modificación de los datos. Permite visualizarlos y aplicar los filtros o modificaciones expuestas en el apartado ***2.1.4 Análisis y Procesamiento de los datos*** de este documento.

3.1.1.3 Módulo de modelos

Este bloque está basado en el concepto de clase abstracta, la cual implementa métodos generales como el entrenamiento, la evaluación o la inserción de los datos a la clase. Los modelos que extienden esta clase solo deberán generar la arquitectura correspondiente.

3.1.2 Código

El código se ha escrito en el lenguaje *Python* versión *3.6*. Además, se ha hecho uso de las siguientes librerías que facilitan el manejo, procesamiento de datos y creación de modelos de aprendizaje automático: *numpy*, *matplotlib*, *tensorflow*, *keras* y *scipy*.

La librería *numpy* facilita el manejo de datos. Está especializa en el tratamiento optimizado de matrices y vectores.

Matplotlib se encarga de dibujar gráficos. Algunos de los mostrados en este trabajo han sido realizados con esta librería.

Tensor*flow* y *keras* trabajan de forma conjunta para la generación de modelos de redes neuronales. La primera se especializa en el cálculo eficiente y paralelo. En caso de tener una tarjeta gráfica de la marca *Nvidia* hará uso de ella. La segunda librería se encarga de generar modelos de redes neuronales como las redes convolucionales, las redes multicapa o las redes recurrentes *LSTM*.

S*cipy* se encarga de tratar señales. Se ha hecho uso de alguna de sus funciones como la *convolución* o la *transformada de Fourier*.

Otras librerías no usadas para las pruebas, pero que son útiles al predecir o clasificar series temporales son *sktime* [25] y *scikit-learn*. La primera, está especializada en el manejo de series temporales. El inconveniente de usar esta librería es que aún se encuentra en una versión muy temprana del desarrollo. La segunda se especializa en la creación y entrenamiento de modelos de aprendizaje automático. Esta permite utilizar modelos como *árboles de decisión* y *KNN* entre otros.

3.2 *Desarrollo*

Para que el análisis arroje datos relevantes, en este apartado se especificarán los criterios y los procedimientos a utilizar, manteniendo los objetivos expuestos en el apartado ***1.2 Objetivos***. Además, se detallarán los datos a utilizados para las pruebas.

3.2.1 Datos de estudio

En este apartado se analizarán los datos con los que se van a realizar las pruebas. En concreto se usarán los datos correspondientes a lenguaje de signos: *LIBRAS* y *uWaveGestureLibrary*. En el ***Anexo A – Dataset clasificación*** se puede encontrar más información sobre cómo se han obtenido estos conjuntos de datos.

La división de los datos para el entrenamiento y test también se especifican en este anexo.

3.2.1.1 LIBRAS

El nombre de *LIBRAS* pertenece a la abreviatura de *"LÍngua BRAsileira de Sinais"* en portugués o *lenguaje de signos brasileño* traducido al castellano. Los datos se han obtenido a partir de videos, donde una persona mueve la mano para realizar distintos signos. Cada signo se asocia a una clase. Del video se han escogido 45 *frames* representativos, en los que los valores de los datos se corresponden a las coordenadas *X* e *Y* donde se encuentra la mano, por lo que el *dataset* cuenta con dos variables. En la *figura 3-2* se muestran 6 ejemplos de datos.

Todos los datos ya han sido normalizados, tal y como se indica en la página donde se obtiene dicho conjunto de datos [26].

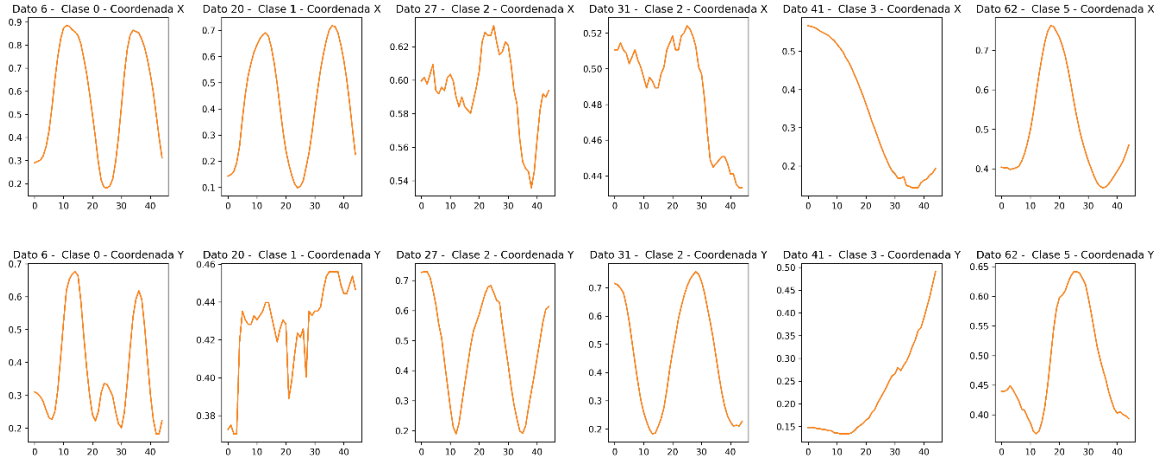

Figura 3-2: Ejemplo de series del conjunto de datos LIBRAS.

(Los ejemplos están etiquetados atendiendo a las coordenadas, el número del dato y la clase)

Como se ha explicado, este *dataset* se corresponde con las coordenadas de una mano cuando realiza un signo. Estos movimientos son cortos, por lo que en las series temporales cada variable contará solamente con 45 datos. A su vez estos datos ya han sido normalizados y los valores se mantienen en el rango de *0* a *1*.

Para analizar la estacionariedad de la serie temporal debemos buscar si se producen repeticiones o ciclos en los datos. También se pueden observar similitudes entre datos de la misma clase. En la *figura 3-3* se observan algunos datos pertenecientes a la clase *1*.

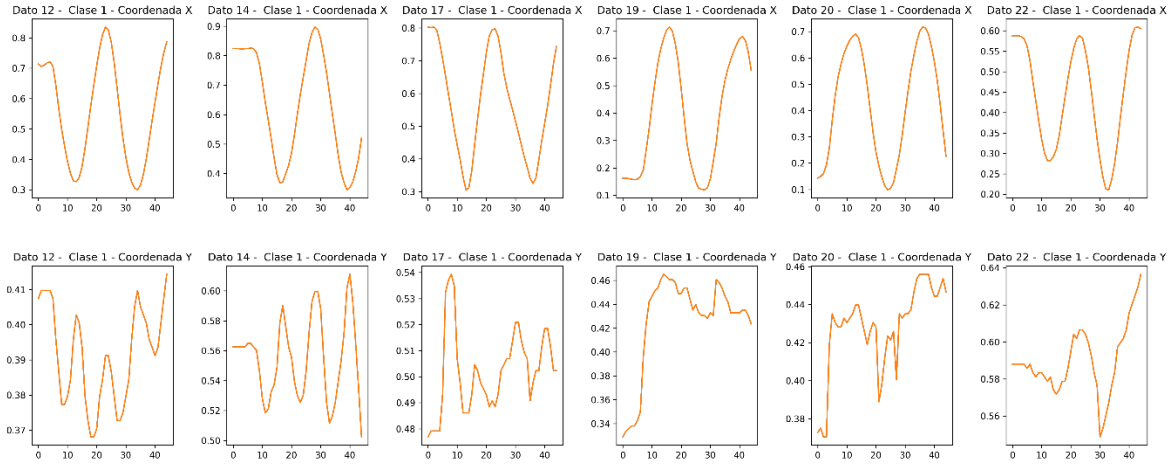

Figura 3-3: Ejemplo datos clase 1 en dataset *LIBRAS*.

(Los ejemplos están etiquetados atendiendo a las coordenadas, el número del dato y la clase)

Como se observa en la figura expuesta, las series temporales son breves y los datos no presentan repeticiones periódicas. Los datos de la misma clase tampoco son iguales, por lo que se concluye que se trata de una serie *no estacionaria*.

Según el análisis de estos datos, podemos entrever que aplicando la normalización no obtendríamos mejora alguna, ya que ha sido previamente aplicada. Por otro lado, la codificación que calcularía la diferencia entre series tampoco debería arrojar buenos resultados, ya que los datos de la misma clase no son iguales y se producirían artefactos como el efecto *offset* explicado en el apartado ***2.1.5.2 Distancia entre series***.

Todas estas hipótesis serán evaluadas en las pruebas.

3.2.1.2 uWaveGestureLibrary

Este conjunto de datos también corresponde al lenguaje de gestos. En este caso se trata de un lenguaje inventado, cuyos gestos se muestran asociados a cada clase en la *figura 3-4* de forma esquemática. Los datos se han obtenido mediante un acelerómetro, colocado en la mano que ha hecho los movimientos. Este sensor medía las coordenadas *X, Y, Z*. En la página [27] se puede encontrar la figura adjunta.

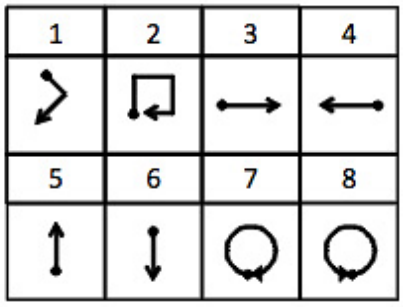

Figura 3-4: Movimientos de manos asociados a cada clase.

Los datos se componen de 3 variables, tal y como se observa en la *figura 3-5*. En este caso, el *dataset* no han sido normalizado, por lo que encontramos valores en un rango dispar. Estos datos tampoco cuentan con ciclos, por lo que se consideran datos *no estacionarios*.

En la *figura 3-6* se muestran algunos de los datos de la clase *1*. Como se observa, los datos son más parecidos los unos a los otros, por lo que la codificación de cálculo de distancia podrá arrojar mejores resultados.

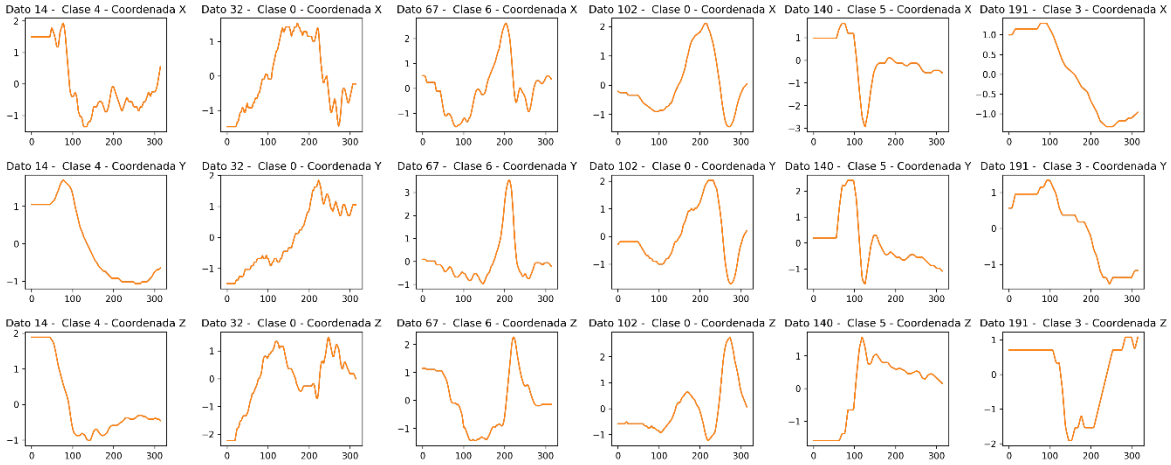

Figura 3-5: Ejemplo de series temporales del conjunto de datos uWaveGestureLibrary.

(Los ejemplos están etiquetados atendiendo a las coordenadas, el número del dato y la clase)

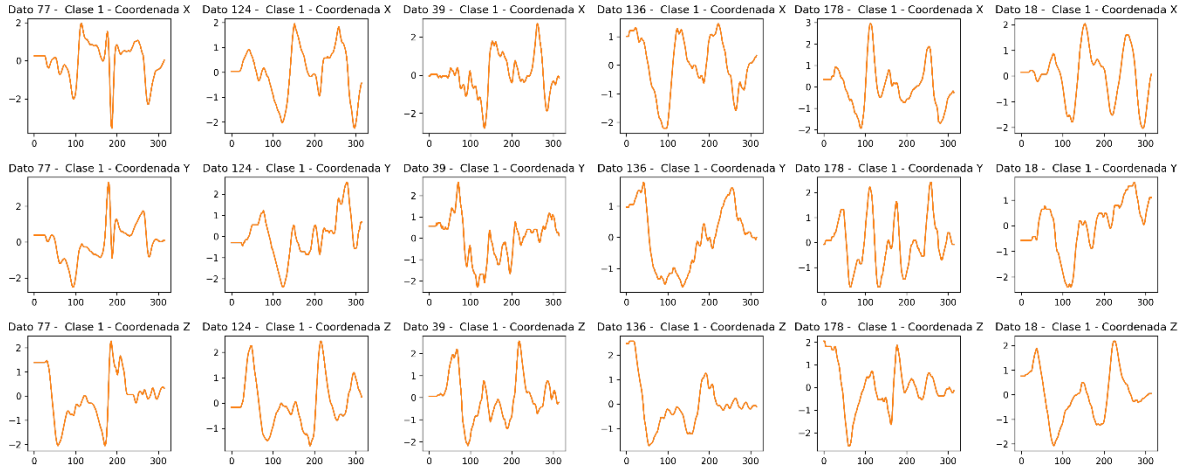

Figura 3-6: Ejemplo datos clase 1 en dataset *LIBRAS*.

(Los ejemplos están etiquetados atendiendo a las coordenadas, el número del dato y la clase)

En este caso la normalización también arrojaría mejores resultados ya que estos datos no han sido normalizados previamente.

3.2.2 Criterios del análisis

Para que los datos sean lo más robustos posibles aplicaremos los siguientes criterios que se deben cumplir en el análisis de resultados:

1. Dado que los modelos de aprendizaje automático utilizados en este TFG incluyen parámetros de inicialización aleatoria, los resultados mostrados se deben obtener con una media de 10 ejecuciones.
2. La resolución de los datos será redondeada a cuatro cifras decimales.
3. Se deberá especificar el número de épocas requeridas para la obtención de los resultados expuestos.
4. Para las pruebas de medición de tiempo será necesario hacer 5 ejecuciones sobre el mismo hardware sin que se estén ejecutando otros programas o aplicaciones. Se tomará el valor medio de los obtenidos en las 5 ejecuciones.

Las métricas que se van a utilizar para el análisis de resultados son:

- La precisión del modelo en la clasificación se medirá por la métrica *accuracy* sobre los datos de prueba. Esta métrica mide porcentualmente, en un rango del *0* al *1*, la proporción de datos que han sido clasificados correctamente. Es decir, que la clase predicha por el modelo es la misma que se les había asignado a estos datos.
- La métrica *ECM* se usará para medir el *loss* o pérdida. *ECM* o error cuadrático medio, se calcula con la fórmula adjunta, donde *n* es el número de datos, $\widehat{Y}_i$ el vector de resultados e *Y* el vector de valores verdaderos del entrenamiento:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}\left(\widehat{Y}_i - Y_i\right)$$

- La medición de tiempo se hará en segundos y siempre se deberá especificar a qué conjunto de datos pertenece cada medición

Se han escogido estas métricas ya que son las más comúnmente utilizadas en el aprendizaje automático. De esta forma se podrán comparar estos resultados con los de otros artículos.

4 Pruebas y resultados

En este apartado se explicarán los resultados de las pruebas, combinando los modelos de aprendizaje automático, mecanismos de preprocesamiento de datos y otras técnicas expuestas en el apartado de ***2. Estado del Arte***.

En concreto se compararán los distintos modelos de aprendizaje automático aplicando diferentes codificaciones y modificaciones a los datos del ***Anexo I***. Se explicarán con detalle los resultados obtenidos, indicando qué codificaciones y modelos de redes neuronales arrojan los mayores valores de precisión en la clasificación de series temporales.

4.1 *Pruebas*

En esta sección se comentarán las pruebas a realizar, detallando cómo se realizan y qué técnicas se aplican. Todos los datos de este apartado serán necesarios para entender el contexto de los resultados.

En primer lugar, indicar que los resultados nombrados como *estándar* en las tablas se refiere a que no se ha aplicado ninguna técnica de codificación sobre los datos, más allá de la necesaria para utilizar la arquitectura correspondiente del modelo de clasificación.

Además, en algunos casos, en las tablas de resultados, se podrá encontrar el símbolo >, el cual hace referencia al orden en el que se han aplicado las codificaciones o técnicas de procesamiento de datos. El símbolo + se refiere a que están adjuntando los resultados de una modificación de los datos a la serie temporal.

4.1.1 Técnicas de preprocesamiento

Las técnicas de preprocesamiento expuestas en este apartado ya han sido explicadas en ***2.4.1 Análisis y Procesamiento de los datos***, por lo que en esta sección solo se detallará la metodología con la que han sido aplicadas.

4.1.1.1 Filtro de media y derivada

El filtrado de media es un filtro muy usado en el preprocesamiento de señales. En este caso haremos uso del *kernel*: *[1, …, 1] / N*, donde el núcleo tiene longitud *N*. En las tablas de resultados aparece el nombre de *media* junto a un número entre paréntesis. Este número se corresponderá con la *N* del tamaño del núcleo aplicado para este filtrado.

La derivada se trata como un filtrado que calcula la diferencia entre datos adyacentes. Consiste en aplicar el *kernel*: *[-1, 1]*. Esto mejorará la representación del tiempo en la serie temporal.

También se ha medido la diferencia entre variables que constituyen los registros de la serie temporal de un dato, la cual que se indica como *diferencia* en las tablas de resultados. Se calcularán tantas diferencias como posibles combinaciones haya entre variables en la serie temporal. Por ejemplo, en una serie con 3 variables *X, Y, Z* se medirían las diferencias entre los distintos registros de X e Y, Y Z, X y Z. Esto añade más información de la estructura de los datos y su relación en la entrada al clasificador.

4.1.1.2 Normalización

Como ya se ha mencionado, existen distintos tipos de normalizaciones. En este caso se buscará dejar la media a *0* y la desviación típica a *1*. Para ello será necesario calcular la media y la desviación típica para cada variable de todos los datos de entrenamiento. Posteriormente, se normalizarán los datos restando la media y dividiendo entre la desviación típica correspondiente de cada variable.

4.1.1.3 Sampleo

El *sampleo* consiste en reducir la resolución de los datos. Para aplicar esta técnica se cogerá uno de cada *N* datos, donde *N* se especificará entre paréntesis en la tabla de resultados.

En el lenguaje de programación utilizado, *Python 3*, solo será necesario indicar la *N* como tercer índice en las listas: $datos = datos[:: N]$

4.1.2 Codificaciones

En este apartado se desarrollan las codificaciones utilizadas para las pruebas, explicando en detalle cómo se han aplicado a los datos *multivariantes*.

4.1.2.1 Distancia

Como se indica en el apartado ***2.1.5.2 Distancia entre series*** esta codificación consiste en calcular la diferencia entre series temporales. Las series de referencia serán los datos de entrenamiento.

La operación de cálculo de distancia se puede realizar por varios métodos. En este caso hemos utilizado la distancia *euclídea* y la distancia *coseno*. Para saber qué método de cálculo se está utilizando en las pruebas se indicará entre paréntesis al lado del nombre de la codificación.

4.1.2.2 Transpuesta

Mediante esta transformación cambiamos la forma en la que la información temporal se presenta al clasificador, de esta manera validamos cómo afecta el no tener en cuenta la estructura temporal de los datos.

4.1.3 Topología de los modelos de aprendizaje automático

En este apartado se explicarán las topologías de los modelos utilizados para realizar las pruebas, atendiendo a la información temporal que presentan. Recordemos que los datos con los que se van a realizar las pruebas son series temporales *multivariantes*, por lo que los modelos deberán analizar múltiples variables que se generan durante el mismo periodo de tiempo.

Los modelos de redes neuronales utilizados para las pruebas son: la red multicapa o *fully connected*, una red recurrente de tipo *LSTM* y una red neuronal convolucional o *CNN*.

Las funciones de activación usadas son *relu* para la capa de entrada y capas ocultas, mientras para la capa de salida se ha utilizado *tanh*.

4.1.3.1 Topología unidimensional

La topología unidimensional surge de la idea de la codificación vectorial. Por consiguiente, todos los datos de la serie temporal deberán ser analizados a la vez. En este apartado se detallarán las topologías de las distintas redes utilizadas durante las pruebas.

La red multicapa presenta una arquitectura en la que todas las capas están conectas con su posterior, por lo que, esta red conectará todas las neuronas de la primera capa con cada uno de los datos. Esto eliminará la estructura temporal de la serie. Por lo tanto, la red solo contará con la capacidad de analizar de forma global los datos, como si se tratara de un problema en el que no se tienen en cuenta los detalles de la asociación temporal. En la *figura 4-1* se muestra un ejemplo esquemático de esta topología.

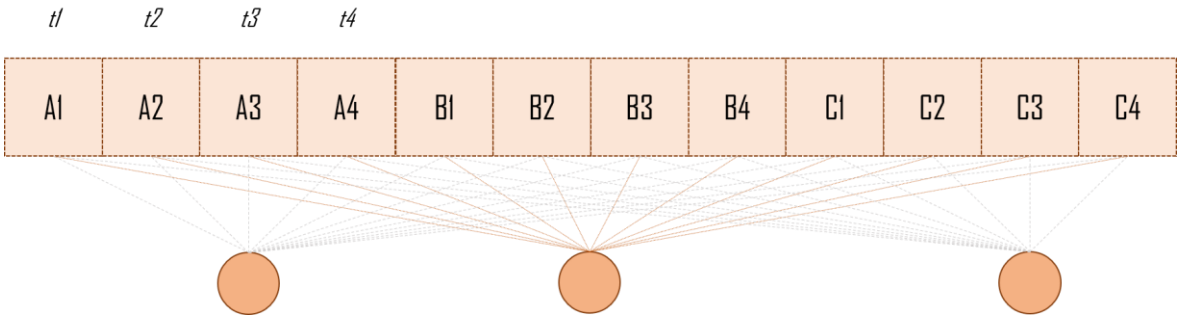

Figura 4-1: Esquema topología unidimensional.

La red recurrente de tipo *LSTM* está diseñada para tener en cuenta el orden temporal de los datos. La neurona *t* transmitirá el resultado del procesamiento del dato X_t a la neurona *t+1*, la cual analizará el dato X_{t+1}. Por consiguiente, se preserva un orden temporal. Debido a la codificación unidimensional, todos los datos deberán ser analizados por una única capa. La topología de la red hará que la neurona *n* (donde *n* es el tamaño de una variable) transmita el resultado de esa neurona a la siguiente, la cual estará procesando datos de otra variable. Por esta razón la topología unidimensional no es la más adecuada para este tipo de redes.

Las *CNN*, como se explica en el apartado ***2.2.1.3 Redes Convolucionales***, aplican convoluciones a los datos, por lo que se genera un efecto de codificación en ventana preservando así la estructura temporal. Así pues, este efecto de ventana permite analizar la evolución de los datos adyacentes, dependiendo del tamaño del *kernel* a aplicar. El inconveniente, es que la red no podrá diferenciar las distintas variables que componen la serie que está analizando.

4.1.3.2 Topología bidimensional

La topología bidimensional se basa en la codificación en matriz. De forma que cada variable será analizada de forma independiente, extrayendo sus características esenciales. A continuación, analizaremos las topologías propuestas para cada tipo de red neuronal.

Las redes multicapa están diseñadas típicamente para hacer uso de datos codificados de forma unidimensional. Para conseguir procesar datos de más de una dimensión se generarán tantas redes multicapa como variables tenga el problema, como se muestra en la *figura 4-2*. Al tratarse de una red *fully connected*, como se comenta en la topología unidimensional, la red no tiene en cuenta los detalles de la estructura temporal de los datos, ya que cada dato estará conectado con todas las neuronas de entrada de la red.

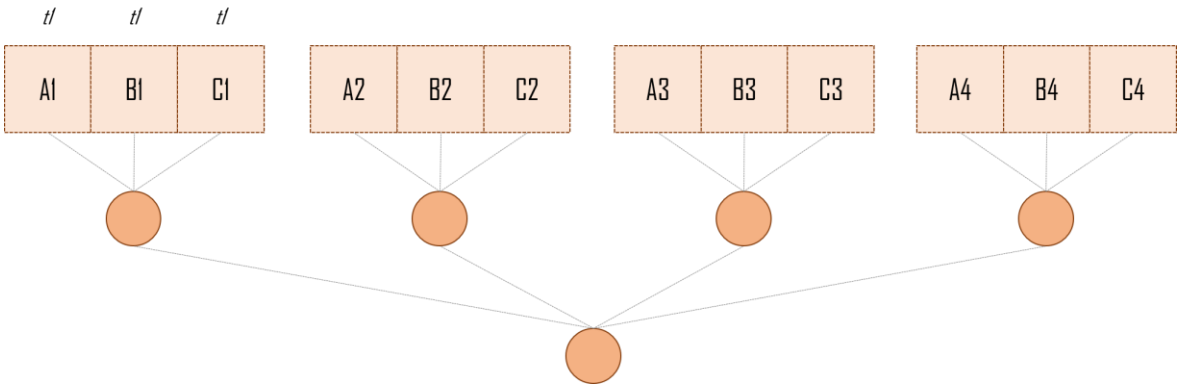

Figura 4-2: Esquema topología bidimensional

La red recurrente de tipo *LSTM*, como ya se ha explicado, mantiene la estructura temporal gracias a su diseño, en el que los resultados de una neurona pasan a la siguiente. Además, en este caso la red recurrente analizará cada variable de forma independiente, extrayendo sus características más relevantes. Esto permitirá a la red diferenciar las variables.

Las redes convolucionales, por su parte, aplican convoluciones sobre los datos. Este tipo de redes admiten entradas multidimensionales, por lo que la entrada de la red será la matriz bidimensional de datos. De esta forma, el modelo aplicará convoluciones con un núcleo de dos dimensiones, permitiendo así analizar la evolución temporal con una codificación en ventana y examinar todas las variables de la serie temporal en el mismo instante de tiempo.

4.2 *Resultados*

Tras la realización de las ejecuciones combinando modelos y distintas codificaciones se han obtenido los resultados que detallamos a continuación.

Cabe destacar que las ejecuciones se han llevado a cabo como se especifica en el apartado ***3.2.2 Criterios del análisis*** y con las metodologías detalladas en el apartado ***4.1 Pruebas***, para llevar a cabo cada codificación y modificación de los datos.

Cada prueba se asociará con las métricas de precisión o *accuracy* y el error cuadrático medio o *ECM*, obtenidos en la clasificación del conjunto de datos de prueba. Además, se remarcan al menos los dos mejores resultados.

En este apartado solo se muestran las tablas de resultados, más adelante se analizarán y compararán.

4.2.1 LIBRAS

Los resultados de las ejecuciones realizadas sobre el *dataset LIBRAS* se muestra en la *tabla 4-1*. Se han hecho pruebas recursivas sobre las codificaciones que arrojaban mejores resultados. En este caso, la precisión más alta se ha obtenido con el modelo *CNN*, con una codificación en serie y aplicando la derivada.

Para obtener estos resultados se han seguido las pautas anteriormente descritas y se han entrenado los modelos durante 300 épocas.

		Fully Connected		LSTM		CNN	
Topología	Codificación / Modificación	Accuracy	*ECM*	Accuracy	*ECM*	Accuracy	*ECM*
Unidimensional	*Estándar*	0,8028	0,0301	0,8439	0,0307	0,8839	0,0260
	Distancia (euclídea)	0,7161	0,0331	0,4272	0,0466	0,8500	0,0266
	Distancia (coseno)	0,8494	0,0245	0,8589	0,0236	0,8706	0,0200
	Estándar + Derivada	0,8444	0,0317	0,8372	0,0396	0,9093	0,0258
	Estándar + Diferencia	0,8433	0,0296	0,8100	0,0303	0,8983	0,0241
	Derivada > Distancia (coseno)	0,8428	0,0229	0,8650	0,0222	0,8872	0,0184
	Media (3)	0,7600	0,0332	0,6733	0,0364	0,8417	0,0279
	Normalización	0,8139	0,0403	0,8372	0,0396	0,8711	0,0238
	Sampleo (3)	0,8000	0,0303	0,7083	0,0367	0,8194	0,0295
Bidimensional	*Estándar*	0,7767	0,0333	0,8039	0,0284	0,8361	0,0298
	Distancia (euclídea)	0,6822	0,0375	0,8150	0,0253	0,8272	0,0227
	Estándar + Derivada	0,8630	0,0287	0,8019	0,0288	0,8648	0,0276
	Estándar + Diferencia	0,8356	0,0303	0,8017	0,0294	0,8450	0,0282
	Transpuesta	0,6361	0,0437	0,7506	0,0395	0,7561	0,0289

Tabla 4-1: Tabla de resultados de la clasificación con el dataset *LIBRAS*

4.2.2 uWaveGestureLibrary

En la *tabla 4-2* se muestran los resultados de las ejecuciones sobre el segundo *dataset*. En este caso encontramos que varias codificaciones y modificaciones de datos arrojan unos resultados parecidos. Por ello se marcan tres resultados como los mejores.

El número de épocas requeridas para que los modelos aprendan con estas tasas de precisión ha sido de 60 épocas.

		Fully Connected		LSTM		CNN	
Topología	Codificación / Modificación	Accuracy	*ECM*	Accuracy	*ECM*	Accuracy	*ECM*
Unidimensional	*Estándar*	0,8675	0,0520	0,8779	0,0418	0,8897	0,0358
	Distancia (euclídea)	0,1249	1,0187	0,3058	0,5636	0,1249	1,0875
	Estándar + Derivada	0,8790	0,0436	0,7906	0,0652	0,8935	0,0377
	Estándar + Diferencia	0,8822	0,0411	0,8692	0,0419	0,8882	0,0382
Bidimensional	*Estándar*	0,8095	0,0610	0,8502	0,0453	0,8994	0,0297
	Distancia (euclídea)	0,4216	0,1395	0,7870	0,0482	0,7655	0,2304
	Distancia (coseno)	0,8439	0,0481	0,8796	0,0294	0,9218	0,0217
	Estándar + Derivada	0,8215	0,0717	0,8584	0,0429	0,8992	0,0299
	Estándar + Diferencia	0,8237	0,0553	0,8549	0,0420	0,9060	0,0289
	Derivada > Distancia (coseno)	0,8793	0,0382	0,8874	0,0298	0,9222	0,0223
	Media (3)	0,6405	0,0902	0,7370	0,0637	0,8807	0,0356
	Normalización	0,7953	0,0620	0,8540	0,0427	0,9024	0,0301
	Sampleo (3)	0,7996	0,0584	0,8704	0,0373	0,9062	0,0274
	Sampleo (3) > Distancia (coseno)	0,8679	0,0364	0,8914	0,0296	0,9230	0,0217
	Sampleo (3) > Derivada > Distancia (coseno)	0,8736	0,0373	0,8843	0,0307	0,9112	0,0238
	Sampleo (3) > Derivada	0,8375	0,0650	0,8659	0,0370	0,9066	0,0265
	Transpuesta	0,7172	0,0669	0,8838	0,0431	0,8767	0,0307

Tabla 4-2: Tabla de resultados de la clasificación con el dataset *uWaveGestureLibrary*

4.3 *Análisis de resultados*

En este apartado se procederá a analizar los resultados obtenidos en las pruebas. Además, se explicará con detalle los motivos por los que se obtienen estos resultados con el fin de conocer qué codificaciones, técnicas de preprocesamiento y modelos de redes neuronales arrojan mayor precisión en la clasificación de las series temporales multivariantes consideradas.

Para ello se hará uso de los dos conjuntos de datos explicados en el apartado ***3.2.1 Datos de estudio***, que corresponden a un problema de lenguaje de signos.

Nota: Los datos de todas las subtablas mostradas en este apartado se han extraído de las tablas de resultados: del dataset LIBRAS se han extraído datos de la topología unidimensional y del dataset uWaveGestureLibrary se han extraído datos de la topología bidimensional.

4.3.1 Análisis de las topologías

En función de sus topologías los modelos de aprendizaje automático analizan los datos de una forma diferente desde el punto de vista de la secuenciación del tiempo y la gestión de la estructura temporal de los datos.

La topología unidimensional consiste en unir todas las variables de una serie temporal en un vector unidimensional para que puedan ser procesadas a la vez por el modelo. Desde el punto de vista teórico esta codificación dificulta al modelo diferenciar unas variables de las otras.

La topología bidimensional es adecuada para acceder fácilmente a toda la estructura temporal de una variable dada. Como se explica en el apartado ***4.1.3.2 Topología bidimensional***, el modelo deberá extraer las características más relevantes de cada variable para posteriormente vectorizarlas y analizarlas conjuntamente.

Examinando los resultados de las ejecuciones para las diferentes topologías, observamos que cada *dataset* obtiene mejores resultados con un tipo distinto de representación de los datos. En el siguiente apartado se estudiará qué modelo de aprendizaje automático presenta mejores resultados.

4.3.2 Análisis de los modelos

En esta sección estudiaremos las razones por las que se obtienen diferentes resultados de precisión para cada modelo de aprendizaje automático. Hay que destacar que las *redes neuronales convolucionales* suelen arrojan mejores resultados que el resto de modelos en las pruebas realizadas.

En la *tabla 4-3* se detallan los resultados obtenidos por cada modelo de aprendizaje automático para los dos *datasets* estudiados, sin aplicar ninguna técnica de preprocesamiento de datos.

	Fully Connected	LSTM	CNN
LIBRAS	0,8028	0,8439	0,8839
uWaveGestureLibrary	0,8095	0,8502	0,8994

Tabla 4-3: Precisión (accuracy) de los modelos sin aplicar ninguna modificación a los datos.

Las redes *fully connected* son redes con múltiples capas, donde todas las neuronas de una capa están conectas con todas las neuronas de la siguiente, lo que dificulta el procesamiento de la secuenciación del tiempo en muchas codificaciones. Al ser redes que no tienen en cuenta la secuenciación temporal de series multivariantes, encontraremos una precisión menor que para el resto de modelos probados.

El modelo *LSTM* es un tipo de *red neuronal recurrente*. Como ya se comenta en la sección ***2.1.1.2 Redes Recurrentes***, este tipo de redes se basan en la posibilidad de recuperar información a partir de datos con errores. Si se está analizando un conjunto de datos, donde los datos de la misma clase tienen patrones similares, está red será capaz de determinar con facilidad a qué clase pertenece este dato. Por ello, en los resultados de las pruebas se obtiene en torno a un 3% de mejora de la precisión respecto a la red multicapa.

Las redes *neuronales convolucionales* fueron diseñadas para el análisis y tratamiento de imágenes. Estas redes aplican convoluciones sobre los datos con un núcleo que se irá desplazando por todo el dato, generando así un efecto de codificación de ventana, como el explicado en el apartado ***2.1.5.1 Enventanado***. Los valores de este *kernel* irán variando a lo largo del tiempo, ya que son los pesos que debe ajustar la red neuronal y representan de esta manera la estructura temporal. Esta hipótesis se confirma a la vista de los resultados, ya que es la red que mejores resultados arroja, con un 3% más de precisión que las redes recurrentes y un 6% más que las redes multicapa.

Con los resultados obtenidos en los distintos modelos de redes neuronales podemos afirmar que las redes convolucionales o *CNN* arrojan unos resultados entre un 3% y 6% mejores que el resto de los modelos usados. Mientras que las redes *fully connected* ofrecen la peor precisión.

4.3.3 Análisis de codificaciones y preprocesamiento de los datos

A continuación, se comentarán detalladamente los resultados de las pruebas aplicando distintas codificaciones y técnicas de preprocesamiento a los datos. Además, se explicarán las razones por las que se dan estos resultados.

4.3.3.1 Codificaciones

En primer lugar, estudiaremos la codificación de distancia entre datos, ya explicada en el apartado ***2.1.5.2 Distancia entre series***. Hemos probado esta codificación aplicando la *distancia euclídea* y la *distancia coseno*. En la *tabla 4-4* se muestran las medias de los resultados obtenidos aplicando esta codificación.

	Fully Connected	LSTM	CNN
Estándar	0,8061	0,8470	0,8916
Distancia (euclídea)	0,5494	0,5799	0,8164
Distancia (coseno)	0,8467	0,8692	0,8958

Tabla 4-4: Precisión (accuracy) al aplicar diferencia entre datos.
(Es la media de los resultados para ambos datasets)

Con la *distancia coseno* conseguimos resultados con una precisión entre un 2% y 3% mayor a la codificación estándar.

En cambio, aplicando la *distancia euclídea*, obtenemos resultados dispares e inferiores a los resultados sin aplicar esta codificación. Esto se puede deber al efecto de *offset* ya comentado. Sucede cuando datos de la misma clase presentan variaciones temporales. Es decir que, a pesar ser patrones iguales, estos patrones se encuentran desplazados en tiempo. Este efecto puede producir que la distancia entre dos datos de la misma clase sea grande.

En el análisis preliminar de los datos, se encontró que el *dataset LIBRAS* presentaba un problema de *offset*. En la *figura 4-3* se muestra un ejemplo con datos de este *dataset* en el que se ve claramente este problema.

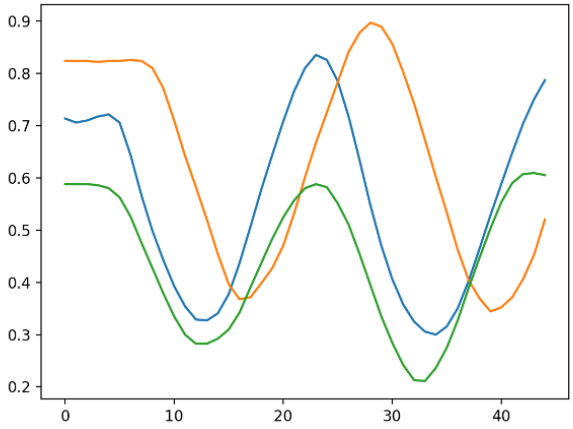

Figura 4-3: Ejemplo offset con datos de la misma clase.

Como vemos, el efecto *offset* se agrava cuando se aplica la distancia euclídea, ya que calcula la diferencia entre datos de forma lineal, al contrario que la distancia coseno, que aplica una normalización. Debido a esto, la distancia coseno suele ser una operación muy usada en el cálculo de distancias vectoriales.

Si aplicamos la transposición de los datos de entrada para forzar la pérdida de la estructura, temporal el resultado de la clasificación empeora, como se muestra en la *tabla 4-5*.

	Fully Connected	LSTM	CNN
Estándar	0,8061	0,8470	0,8916
Matriz transpuesta	0,6754	0,8144	0,8142

Tabla 4-5: Precisión (accuracy) de la codificación transpuesta
(Es la media de los resultados para ambos datasets)

4.3.3.2 Procesamiento de datos

Para conseguir mejorar los resultados también se pueden aplicar pequeñas modificaciones a los datos. En las pruebas realizadas se ha calculado la media y la derivada, se ha modificado la resolución de sampleo y normalizado los datos. En la *tabla 4-6* se han recopilado los resultados.

Como se muestra en esta tabla, al calcular la derivada a los datos se obtiene una mejora en la precisión respecto a los datos estándar de en torno al 3%. La derivada en series temporales consiste en comparar un dato con su predecesor. Esto mejora la relación entre datos de una misma señal, ofreciendo una mejor codificación del tiempo.

Al calcular la diferencia, se asocia todas las variables, como ya se ha explicado en el apartado ***4.1.1.1 Filtro de media y derivada***. Esta asociación entre variables añade más información sobre la estructura de los datos, lo que aumenta la precisión entorno un 2% respecto los datos estándar.

	Fully Connected	LSTM	CNN
Estándar	0,8135	0,8436	0,8769
Derivada	0,8329	0,8478	0,9042
Diferencia	0,8335	0,8322	0,9021
Media (3)	0,6977	0,7044	0,8610
Normalización	0,8046	0,8456	0,8866
Sampleo (3)	0,7998	0,7852	0,8617

Tabla 4-6: Medias resultados procesamiento de datos (Es la media de los resultados para ambos datasets)

Con el filtrado de media conseguimos suavizar picos o variaciones en una señal. En caso de tener mucho ruido esta operación podría ser muy útil y arrojar valores de precisión mayores. Sin embargo, si los datos son muy claros, como los de estos *dataset*, se podrían eliminar detalles relevantes que impedirían la correcta clasificación de las series temporales.

La normalización, como se explica en el capítulo ***2.1.4.2.3 Normalización***, trata de modificar la escala de los datos. En la tabla de ejecuciones se observa que normalizar los datos aporta una pequeña mejora respecto a no aplicar ningún procesamiento. En el caso del *dataset LIBRAS* la normalización ya ha sido aplicada, por lo que no se consigue ninguna mejora, tal y como se explicó en el análisis de datos.

En vista de los resultados, el sampleo o modificación de la resolución de los datos, empeora la precisión en la clasificación de las series. Esto se debe a que la resolución de los datos ha sido previamente ajustada. Esta técnica de procesamiento no suele arrojar una mejora notablemente sobre la precisión, ya que su principal cometido es reducir el tamaño de los datos y así conseguir reducir los tiempos de entrenamiento de los modelos.

4.3.3.3 Combinación de codificaciones y modificaciones

En inteligencia artificial se suelen combinar varias técnicas de procesado de datos para obtener unos mejores resultados en la clasificación. En las pruebas realizadas se han combinado aquellas técnicas que habían arrojado mayores valores de precisión.

En la *tabla 4-7* se muestran los resultados obtenidos con el conjunto de datos *LIBRAS*. En este caso, se ha probado a calcular la derivada y, sobre esta modificación de datos, codificar con la distancia coseno. Si bien con esta combinación no se superan los resultados máximos obtenidos por la *red convolucional*, sí que se han obtenido unos mejores resultados que la aplicando la codificación de diferencia coseno sobre los datos estándar.

	Fully Connected	LSTM	CNN
Estándar	0,8028	0,8439	0,8839
Distancia (coseno)	0,8494	0,8589	0,8706
Estándar + Derivada	0,8444	0,8372	0,9093
Derivada > Distancia (coseno)	0,8428	0,8650	0,8872

Tabla 4-7: Resultados combinados dataset *LIBRAS*.

	Fully Connected	LSTM	CNN
Estándar	0,8675	0,8779	0,8897
Distancia (coseno)	0,8439	0,8796	0,9218
Estándar + Derivada	0,8215	0,8584	0,8992
Derivada > Distancia (coseno)	0,8793	0,8874	0,9222
Sampleo	0,7996	0,8704	0,9062
Sampleo > Distancia (coseno)	0,8679	0,8914	0,9230
Sampleo > Derivada > Distancia (coseno)	0,8736	0,8843	0,9112
Sampleo > Derivada	0,8375	0,8659	0,9066

Tabla 4-8: Resultados combinados dataset *uWaveGestureLibrary*.

En la *tabla 4-9* se detallan los resultados de combinar varias codificaciones y técnicas de preprocesamiento de datos sobre el *dataset uWaveGestureLibrary*. A continuación, se analizarán los resultados y explicará la realización de estas pruebas.

La combinación de calcular la derivada y posteriormente codificar estos datos con la distancia coseno arroja mejores resultados. Esto se puede deber a que la codificación de distancia ya tenía valores de precisión elevados. Al calcular la derivada, como ya se ha comentado en numerosas ocasiones, se aporta información sobre la tendencia de evolución de las variables. Esto ofrece más información a la codificación de diferencia entre series y permite así aumentar la precisión.

En este caso, la modificación de la resolución de los datos parece mejorar la precisión. Esto se debe a que hay cierta información que no es relevante, la cual se puede eliminar sampleando los datos. Al reducir la resolución y posteriormente calcular la diferencia entre series, se consigue aumentar la precisión.

El resto de las combinaciones propuestas arrojan peores resultados que los comentados, debido a que la derivada está calculando la diferencia entre datos adyacentes y al reducir la resolución haya información espacial que se pierde y que era relevante para el cálculo de la derivada.

Podemos concluir que es necesario realizar combinaciones de codificaciones y técnicas de preprocesamiento de datos mejora la precisión de nuestro modelo. Aunque sería necesario realizar numerosas pruebas para determinar qué combinaciones de técnicas nos proporcionarán los mayores valores de precisión.

4.3.4 Análisis del rendimiento de los modelos

Un factor determinante a la hora de decidir qué modelo de aprendizaje automático usar es el tiempo que tarda cada modelo en ser entrenado y en clasificar. Por ello, en esta sección se estudiará el impacto de la aplicación de distintas modificaciones a los datos en la ejecución de cada uno de los modelos estudiados.

En la *tabla 4-9* se muestran los tiempos de entrenamiento (entrenando durante 60 épocas con los datos de *train*) y clasificación (sobre todos los datos de test) de los modelos sobre el conjunto de datos *uWaveGestureLibrary*. Los tiempos se miden en segundos, como se indica en el apartado ***3.2.2 Criterios del análisis***.

		Fully Connected		LSTM		CNN	
Topología	Codificación / Modificación	Entrenamiento	Clasificación	Entrenamiento	Clasificación	Entrenamiento	Clasificación
Unidimensional	*Estándar*	6,5680	0,0512	614,3146	7,5641	81,5935	0,9205
Bidimensional	*Estándar*	5,9295	0,0600	11,2412	0,1145	25,9216	0,3031
	Distancia (coseno)	6,5438	0,0589	11,4114	0,1205	19,3096	0,2341
	Estándar + Derivada	6,4609	0,0677	14,6022	0,1694	28,2571	0,3613
	Sampleo (3)	5,3845	0,0471	10,9591	0,1083	14,0570	0,1620

Tabla 4-9: Tiempos medios de ejecución de los modelos (en segundos)

El modelo *fully connected* es el que menos tiempo tarda en entrenar y clasificar. Sin embargo, como se ha comprobado en las tablas de precisión, es el que menor precisión arroja en la clasificación del dataset *uWaveGestureLibrary*.

El modelo *LSTM* con la topología unidimensional es el que más tarda en entrenar y clasificar. Está diseñado para mantener la secuenciación del tiempo y ejecuta las neuronas de manera secuencial, por lo que resulta difícil de paralelizar. Este modelo con una topología bidimensional tarda entre 2 y 3 veces más que las redes multicapa.

La *red convolucional* con topología bidimensional es el modelo que mejores resultados de precisión obtiene para este *dataset*, pero tarda hasta 5 veces más que la red multicapa.

En caso de tener una tarjeta gráfica *Nvidia*, la librería *TensorFlow*, utilizada para crear los modelos, haría uso de los *Tensor Cores* y *Cuda Cores*, lo que reduciría los tiempos de entrenamiento y clasificación.

Al aplicar técnicas de preprocesamiento o modificar la codificación de los datos se están alterando los valores de entrada al modelo, por lo que esto también afectará al tiempo de ejecución de los modelos.

La derivada aumenta la asociación temporal, ya que aporta información del cálculo de la diferencia entre datos adyacentes de cada variable a la serie temporal, aumentado así el número total de datos de la serie. Como se observa en los resultados, también aumenta el tiempo de entrenamiento y clasificación.

Por otro lado, reducir la resolución mediante sampleo disminuye obviamente los tiempos de ejecución, ya que el sampleo reduce el número de datos de la serie temporal.

Dependiendo de la codificación de los datos o el hardware con el que se cuente, los tiempos pueden variar enormemente. Por ello se recomienda que antes de escoger un modelo de aprendizaje automático se hagan pruebas con un fragmento del conjunto de datos sobre el hardware que se va a usar, con el fin de medir tiempos y precisión obtenidos en cada caso.

5 Conclusiones y trabajo futuro

5.1 *Conclusiones*

El análisis de datos asociados al tiempo es un problema común, que pocas veces se aborda desde el punto de vista de la caracterización precisa de la estructura temporal. Por su complejidad, en ocasiones no se le da la relevancia que merece.

Hemos comprobado que el análisis de las características temporales de los datos es una parte fundamental para entender cómo se deben codificar o qué técnicas de preprocesamiento se deben aplicar para mejorar la representación de su estructura temporal en un clasificador. Tras analizar los conjuntos de datos con los que se iban a realizar las pruebas, hemos hecho varias suposiciones sobre qué resultados obtendríamos al aplicar ciertas técnicas. Al examinar los resultados de las pruebas se han confirmado nuestras hipótesis.

Aunque los dos conjuntos de datos utilizados para las pruebas corresponden al lenguaje de signos, los resultados obtenidos para cada *dataset* son diferentes. Esto se debe, entre otras cosas, a que las técnicas de recolección de los datos fueron distintas y, por tanto, también su estructura temporal. Por ejemplo, en el *dataset LIBRAS*, hemos visto como la codificación de *diferencia entre series* empeoraba la precisión del sistema, debido a que se producía el efecto de *offset*.

A pesar de que con la codificación de distancias en el *dataset LIBRAS* se obtienen malos resultados, en el conjunto de datos de *uWaveGestureLibrary* los resultados son buenos. Por lo que, en general, si los datos tienen una buena alineación temporal, esta codificación mejora la precisión del algoritmo en torno al 2%.

También se ha comprobado que, dependiendo de la codificación, puede ser necesario modificar la topología del modelo. Hay modelos que no soportan datos con más de una dimensión. Será necesario tener en cuenta este hecho a la hora de decidir qué modelo de aprendizaje automático usar.

Aplicando técnicas de preprocesamiento que mejoran la representación de la estructura temporal de las series temporales también puede aumentar la precisión de la clasificación. Por ejemplo, al normalizar, podemos ver una pequeña mejora en los resultados. Otras técnicas, como el filtrado, permiten mejorar la clasificación. El cálculo de la derivada mejora los resultados entre un 2% y 3%; esto se debe a que permite informar sobre la tendencia de la evolución de los datos en el tiempo.

El mejor modelo estudiado son las redes convolucionales que ofrecen una mejora de la precisión de hasta un 6% respecto a los otros modelos probados. En cambio, las redes multicapa son las que peor precisión ofrecen para el conjunto de datos estudiado, aunque es el modelo que menos tarda en ser entrenado de los explorados.

Podemos concluir que, en el contexto de la complejidad del tratamiento de series temporales, cualquier aspecto que contribuya a representar mejor las características temporales de las series puede mejorar su clasificación. Hemos ilustrado este punto aplicando técnicas como la diferencia entre series o la incorporación de la información de la derivada mejorando los resultados en torno a un 3% en los conjuntos de datos utilizados. Es probable que para series temporales más largas que las correspondientes al lenguaje de signos esta mejora sea superior. Otro factor determinante en la precisión de los resultados son los modelos de aprendizaje automático. En este caso, hemos visto como las redes neuronales convolucionales mejoran la precisión hasta un 6% con respecto a otras redes, de nuevo

dentro del contexto de los datos utilizados. Por lo que a pesar de que la codificación del tiempo sea compleja, aplicando técnicas similares a las exploradas que proporcionen mayor información sobre la estructura temporal, se podría mejorar notablemente la precisión en un problema de series temporales multivariantes.

5.2 *Trabajo futuro*

Este trabajo supone un análisis de la clasificación de series temporales multivariantes aplicando redes neuronales. Una aproximación parecida podría utilizarse en los problemas de predicción de series temporales, y en concreto en el contexto de la interpretación de la lengua de signos. Otro contexto en el que se podrían aplicar estas técnicas es el de la predicción de la evolución de enfermedades infecciosas como el *COVID-19*. Sobre este conjunto de datos se podrían obtener resultados que ayudasen a elaborar un modelo de evolución del virus atendiendo a factores críticos en la evolución temporal de la pandemia.

Debido a que el trabajo se ha centrado en el análisis de redes neuronales, sería interesante probar otros modelos como los bosques aleatorios o *random forest* con la codificación *SMTS*, que preserva la estructura temporal [22].

Habiendo tratado datos sobre el lenguaje de signos, se podrían ampliar los conocimientos en este campo y hacer un traductor, como el propuesto en la introducción, optimizado en la dirección explorada en este TFG. Este sería un proyecto complejo, pero podría ayudar a personas con discapacidad auditiva a comunicarse con gente que no conoce el lenguaje de signos.

Existen innumerables proyectos a realizar con las series temporales. El tiempo pocas veces se tiene en cuenta y, como han arrojado los resultados, cuando se codifican los datos o se utilizan los modelos adecuados que preservan la estructura temporal de las series se pueden obtener unos mejores resultados.

Referencias

[1] A. Abanda, U. Mori, and J. A. Lozano, "A review on distance based time series classification," *Data Min. Knowl. Discov.*, vol. 33, no. 2, pp. 378–412, Mar. 2019, doi: 10.1007/s10618-018-0596-4.

[2] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 917–963, Jul. 2019, doi: 10.1007/s10618-019-00619-1.

[3] G. C. Reinsel, *Elements of multivariate time series analysis*. Springer Science & Business Media, 2003.

[4] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *Motion-based recognition*, Springer, 1997, pp. 227–243.

[5] H. Cooper, B. Holt, and R. Bowden, "Sign language recognition," in *Visual Analysis of Humans*, Springer, 2011, pp. 539–562.

[6] D. Rozado, F. B. Rodriguez, and P. Varona, "Extending the bioinspired hierarchical temporal memory paradigm for sign language recognition," *Neurocomputing*, vol. 79, pp. 75–86, 2012, doi: https://doi.org/10.1016/j.neucom.2011.10.005.

[7] M. J. Cheok, Z. Omar, and M. H. Jaward, "A review of hand gesture and sign language recognition techniques," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 1, pp. 131–153, 2019.

[8] A. JAIN, "Complete guide to Time Series Forecasting (with Codes in Python)," Feb. 06, 2016. https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/.

[9] H. Kantz and T. Schreiber, *Nonlinear time series analysis*, vol. 7. Cambridge university press, 2004.

[10] C. W. J. J. Granger, "Some properties of time series data and their use in econometric model specification," *J. Econom.*, vol. 16, no. 1, pp. 121–130, 1981, doi: 10.1016/0304-4076(81)90079-8.

[11] A. K. Wagner, S. B. Soumerai, F. Zhang, and D. Ross-Degnan, "Segmented regression analysis of interrupted time series studies in medication use research," *J. Clin. Pharm. Ther.*, vol. 27, no. 4, pp. 299–309, Aug. 2002, doi: 10.1046/j.1365-2710.2002.00430.x.

[12] K. Mizumoto and G. Chowell, "Transmission potential of the novel coronavirus (COVID-19) onboard the diamond Princess Cruises Ship, 2020," *Infect. Dis. Model.*, vol. 5, pp. 264–270, Jan. 2020, doi: 10.1016/j.idm.2020.02.003.

[13] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas, "Uncertain Time-Series Similarity: Return to the Basics," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1662–1673, 2012, doi: 10.14778/2350229.2350278.

[14] N. E. Huang *et al.*, "The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis," *Proc. R. Soc. London. Ser. A Math. Phys. Eng. Sci.*, vol. 454, no. 1971, pp. 903–995, Mar. 1998, doi: 10.1098/rspa.1998.0193.

[15] "predecir | Diccionario panhispánico de dudas | RAE - ASALE." https://www.rae.es/dpd/predecir.

[16] R. J. Hyndman and G. Athanasopoulos, *Forecasting : principles and practice*. .

[17] R. W. Hamming, *Digital filters*. Courier Corporation, 1998.

[18] H. Wang, J. Wu, P. Zhang, and Y. Chen, "Learning Shapelet Patterns from Network-based Time Series," 2018, doi: 10.1109/TII.2018.2885700.

[19] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019, doi: 10.1186/s40537-019-0197-0.
[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[21] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, “Deep learning for time series classification: a review,” *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 917–963, 2019, doi: 10.1007/s10618-019-00619-1.
[22] M. G. Baydogan and G. Runger, “Learning a symbolic representation for multivariate time series classification,” *Data Min. Knowl. Discov.*, vol. 29, no. 2, pp. 400–422, 2014, doi: 10.1007/s10618-014-0349-y.
[23] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015, doi: 10.1038/nature14539.
[24] M. G. Baydogan and G. Runger, “Learning a symbolic representation for multivariate time series classification,” *Data Min. Knowl. Discov.*, vol. 29, no. 2, pp. 400–422, Feb. 2014, doi: 10.1007/s10618-014-0349-y.
[25] M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, “sktime: A Unified Interface for Machine Learning with Time Series,” in *Workshop on Systems for ML at NeurIPS 2019*.
[26] “UCI Machine Learning Repository: Libras Movement Data Set.” https://archive.ics.uci.edu/ml/datasets/Libras+Movement.
[27] “Time Series Classification Website.” http://www.timeseriesclassification.com/description.php?Dataset=UWaveGestureLibrary.
[28] “Multivariate Time Series Classification with Learned Discretization.” http://www.mustafabaydogan.com/multivariate-time-series-discretization-for-classification.html.

Glosario

LSTM:	Long short-term memory.
CNN:	Convolutional Neural Network (Red Neuronal Convolucional).
RNN:	Recurrent Neural Network (Red Neuronal Recurrente).
Raw:	Sin editar, sin procesar.
ECM:	Error cuadrático medio.
Dataset:	Conjunto de datos.
Train:	Entrenamiento.
Test:	Pruebas.
Kernel:	Núcleo, referido a filtrados o convoluciones.
Accuracy:	Precisión: Porcentaje de datos acertados en la clasificación de modelos de aprendizaje automático.
Loss:	Pérdida: Error cuadrático medio de los resultados de la clasificación o predicción en modelos de aprendizaje automático.

Anexos

A Dataset clasificación

El *dataset* a usar se ha obtenido de la página de *Mustafa Baydogan* [28]. En esta web encontramos varios conjuntos de datos, además del código de un *random forest* escrito en *R*. Toda la información de como descargar este material está en la propia página y tiene un libre acceso.

También nos encontramos con la *tabla 0-1* donde se especifican las características de los *datasets*. Información que podrá ser relevante a la hora de escoger los datos con los que realizar pruebas.

	# of classes	# of variables	Dataset Size Length	Train	Test	CV	Source
AUSLAN	95	22	45-136	1140	1425	10-fold	
Pendigits	10	2	8	300	10692		
Japanese Vowels	9	12	7-29	270	370		
Robot Failure							
LP1	4	6	15	38	50	5-fold	UCI
LP2	5	6	15	17	30		
LP3	4	6	15	17	30		
LP4	3	6	15	42	75		
LP5	5	6	15	64	100		
ECG	2	2	39-152	100	100	10-fold	Olszewski
Wafer	2	6	104-198	298	896		
CMU_MOCAP_S16	2	62	127-580	29	29	10-fold	CMU MOCAP
ArabicDigits	10	13	4-93	6600	2200	x	
CharacterTrajectories	20	3	109-205	300	2558	x	UCI
LIBRAS	15	2	45	180	180	x	
uWaveGestureLibrary	8	3	315	200	4278	x	UCR
PEMS	7	963	144	267	173	x	UCI
KickvsPunch	2	62	274-841	16	10	x	CMU MOCAP
WalkvsRun	2	62	128-1918	28	16	x	
Network Flow	2	4	50-997	803	534	x	Sübakan et al.
DigitsShape	4	2	30-98	24	16	x	
Shapes	3	2	52-98	18	12	x	

Tabla 0-1: Especificaciones de los datos